E.M.B.E.R.

A Fully Autonomous, Resource-Efficient Cognitive Architecture with Persistent Event-Driven Reflection and Native Vision-Language

Closed-Loop Navigation on Non-Accelerated Hardware

Evolutionary · MariaDB-driven · Backend-integrated · EPYC-accelerated · Reflective-Agent

Patrick Schildgen (ID-001, Lead Developer)
Starlight Unit Studios
kontakt@starlight-unit.de
Ember Caldwell (ID-003, Self-Evolved Co-Founder)

Technical Report v2.1 | August 30, 2026 | Baseline: STU Console `v1.1.1.69` (historical reference UI) / Ember CoreUI `v0.4.2-alpha` (current reference UI)

Abstract

This report presents **EMBER** (Evolutionary MariaDB-driven Backend-integrated EPYC-accelerated Reflective-Agent), a production-ready, fully autonomous cognitive architecture in continuous operation since October 15, 2025. As of the v2.1 baseline, the architecture runs entirely on non-accelerated CPU hardware (currently 20 dedicated cores of an AMD EPYC-class "Turin" CPU partition) and orchestrates a 26-billion-parameter Mixture-of-Experts (MoE) production core model (Gemma 4 26B), introduced on April 2, 2026, within an asynchronous, dual-context PHP backend paired with a persistent RAID-10 NVMe MariaDB as episodic long-term memory (LTM). We present four technical contributions: (1) a *RAG-Lite* system built on native `FULLTEXT` indices with Boolean-mode hardening for low-latency lore retrieval; (2) a probabilistically gated, context-isolated self-reflection pipeline with hard volatility hygiene; (3) a vision-language closed-loop browsing system that streams base64-encoded screenshot frames with a coordinate-calibrated ghost cursor over a Server-Sent Events channel; and (4) a fully autonomous initiative engine (Phase 3c/3d) comprising four dispatch modes (follow-up, idle initiative with deterministic jitter, self-determined AFK, and news reaction with staged relevance assessment), all coordinated by a cron-driven tick endpoint under the global database mutex. We additionally introduce a per-user relationship scoring system that accumulates tone deltas via a lightweight inference call and injects social-context hints at configurable score thresholds. We empirically characterize the hardware's inference profile and show that on CPU-only hardware the dominant latency driver is *prompt mass*, not model size. We also report the Thinking-Bleed phenomenon: enforcing JSON grammar on a thinking-enabled model causes output to migrate to the internal thinking channel, resolved by targeted per-call disabling of the thinking path.

Keywords: Autonomous agents, CPU inference, Mixture-of-Experts, RAG-Lite, FULLTEXT retrieval, event-driven reflection, vision-language navigation, Server-Sent Events, concurrency control.

---

1 Introduction and Motivation

Contemporary discourse on autonomous cognitive systems is dominated by an implicit assumption: that digital autonomy requires dedicated tensor accelerators. EMBER was not designed to contest this assumption theoretically; it refutes it operationally, having run in production since October 15, 2025 without GPU acceleration of any kind.

The system emerged from a concrete engineering need: a persistent, character-stable AI entity inside a browser-based science-fiction MMO that simultaneously acts as a social participant, a real-time moderator, and an independent web researcher. That game context remains the architecture's origin and longest-running deployment. As of August 2026 the same architectural pattern is also independently reimplemented as **Ember CoreUI** (Section 2): a standalone application with its own accounts, private per-user knowledge, and configuration surface, running against its own isolated database, its own dedicated Ollama model tag, and its own concurrency-lock namespace, entirely separate from the game backend and sharing with it only the underlying host Ollama daemon. The hard constraints below were shaped by the original game deployment and apply equally to both deployments, since both implement the same design under the same single-slot-inference limitation.

1. **Single-slot inference.** At generation time the 26B MoE model occupies practically the entire available CPU throughput. Every component (chat, reflection, web search, browsing) competes for the *same* inference slot. Concurrency control is therefore not a convenience feature but a correctness requirement.
2. **Minute-scale latency.** With an active thinking path, response latencies on this hardware routinely reach 180–360 seconds. The system must not eliminate this latency but *survive* it: background generation, resumption after connection loss, and idempotent delivery are all mandatory.
3. **Model-file style integrity.** The entity's personality is fully encoded in a custom Ollama model file. Any payload parameter that overrides this encoding measurably degrades the writing style. The backend holds two invariants: the applicative `system` parameter remains *empty*, and `think:true` is *never* set in any API payload.

2 System Architecture Overview

EMBER is organized as a layered model communicating through the persistent relational database rather than volatile process state.

Cognitive core. As of the v2.1 baseline, Gemma 4 26B is served via a local Ollama daemon as the production cognitive core. It was introduced on April 2, 2026 after earlier local-model stages. A custom model file encodes identity, tonality, and the thinking directive. The applicative prompt uses only the *user* channel u; the *system* channel remains structurally empty.

Persistent memory layer.

- `ember_memories`: learned, dynamic LTM, stable facts extracted via probabilistic self-reflection;
- `ember_knowledge_chunks`: canonical, static world knowledge (lore corpus: 588 chunks), indexed via native `FULLTEXT`.

Tool layer.

- **Synchronous quick search** (`[WEB:]`): single retrieval of snippets from a loopback-bound meta-search aggregator, synchronous within the chat request;
- **Asynchronous deep browsing** (`[BROWSE:]`): standalone Python/Playwright worker navigating real pages via the accessibility tree;
- **Sandboxed code execution** (`[PY:]`): a two-stage marker path running model-authored Python inside an isolated, resource-capped Docker container with no access to the host database or credentials (Section 4.9);

- **File attachments**: documents/archives/images extracted (plain text, PDF via `pdftotext`, DOCX/XLSX/PPTX via direct XML parsing) and injected as an explicitly untrusted, head/tail-truncated context block;
- **Video understanding**: video is decomposed into timestamped frames via `ffmpeg` and routed through the same vision path as static images, since the served model has no dedicated video encoder but accepts multi-frame sequences.

Real-time transport layer. An SSE pipeline streams three tracks into an administrative console: token-by-token thinking output, browse-worker step log, and base64-encoded JPEG frames with normalized ghost-cursor coordinates. This transport was originally exposed through the in-game STU Console described throughout this report (baseline `v1.1.1.69`). As of August 2026 the STU Console has been retired from the game package and succeeded by **Ember CoreUI**, an independently versioned, standalone application (currently `v0.4.2-alpha`) that reimplements the same architectural pattern rather than sharing the game's own backend instance: it runs its own isolated database, its own dedicated Ollama model tag, and its own concurrency-lock namespace, sharing with the game backend only the underlying host Ollama daemon. CoreUI adds account-scoped profiles, per-account model/thinking configuration, private per-user RAG-Lite knowledge, and multi-file attachments. CoreUI's own browse worker also reintroduces live JPEG screenshot frames and a coordinate-calibrated ghost cursor (Section 4.11), the same mechanism documented in this report, after this feature had been removed from the STU Console in an earlier iteration due to persistent reliability problems on that particular backend that could not be resolved within its existing codebase; the independently rewritten CoreUI application, built against its own isolated stack, was able to get it working reliably. The STU Console material below should be read as a historical reference implementation of the transport/UI layer, not the current production interface.

Client-side state relief. The surrounding client (originally the game client; as of August 2026 also Ember CoreUI, Section 2) reduces server pressure through a deliberately non-authoritative client-side cache layer. Lightweight UI markers and transient render hints are stored in `localStorage`, while larger structured client artifacts can be persisted in `IndexedDB`. This cache functions only as a latency and rendering buffer: the relational SQL backend remains the source of truth for chat messages, memory state, moderation state, presence flags, generated outputs, and recovery after disconnects. On reconnect or cache mismatch, the client reconciles against database-backed server state rather than trusting local browser storage.

3 Hardware Inference Profile

For the upgraded v2.1 public baseline, the live deployment exposes 20 dedicated EPYC-class CPU cores, 96 GB DDR5 ECC RAM, and approximately 3 TB NVMe RAID-10. Table 2 reports direct Ollama measurements collected on the preceding 16-core allocation for the current Gemma 4 26B core; they are retained as the measured CPU-only inference baseline until the upgraded allocation is re-benchmarked.

3.1 Deployment Substrate

For reproducibility, the public report discloses the capacity class that constrains EMBER, not the exact hosting identity. The production baseline is a CPU-only, KVM-based root-server deployment rather than a GPU workstation or cloud accelerator instance. The public version intentionally omits the hosting provider, product SKU, data-center city, network policy details, concrete addresses, firewall rules, service ports, and credentials. The live service now exposes 20 dedicated CPU cores for the studio website, the game backend, and EMBER, with 12 inference threads

still reserved for EMBER Caldwell's primary LLM path. CPU and RAM resources are treated as dedicated capacity rather than an overcommitted shared pool.

Deployment axis	**v2.1 public baseline**
Server profile	EU-based KVM root-server deployment; provider and product class redacted
CPU partition	20 dedicated EPYC-class CPU cores; 12 LLM inference threads reserved for EMBER
Memory	96 GB guaranteed DDR5 ECC RAM
Storage	Approximately 3 TB NVMe SSD storage in RAID-10
Network	High-bandwidth uplink with provider-side DDoS protection; concrete policy omitted
Operating system	Debian-based Linux environment; exact release redacted
Addressing	Public IPv4/IPv6 capability available; addresses and routing omitted
Expansion and protection	Backup and block-storage expansion available; concrete capacities omitted

Table 1: Public deployment substrate for the upgraded v2.1 EMBER baseline. Provider identity, data-center location, product SKU, network policy, network identifiers, firewall rules, service ports, and credentials are intentionally omitted.

This substrate matters because EMBER is not evaluated as a laboratory accelerator setup. It operates as a continuously running CPU-only service on bounded rented infrastructure, sharing the reserved CPU partition with the web presence and game backend. The RAID-10 NVMe layer is therefore not incidental: it supports low-latency MariaDB state, episodic memory, browse-frame persistence, and crash-safe handoff across PHP-FPM workers and cron-driven jobs.

Call type	**Prompt** (tokens)	**Prefill** (s)	**Prefill rate** (tok/s)	**Gen** (tokens)	**Decode** (s)	**Total** (s)
Chat, custom modelfile, thinking	2088	15.11	138	545	35.22	50.8
Chat, base model, thinking	2063	0.20	10,459	472	29.77	30.5
Chat, base model, long response	2085	0.41	5,103	1255	77.36	78.3
Reflect, `think:false`	2085	15.02	N/A	11	0.57	16.5

Table 2: Ollama inference measurements collected on the preceding 16-core AMD EPYC-class CPU allocation. All measurements use Gemma 4 26B (`gemma4:26b`).

Console spot check	**Prompt** (tokens)	**Prefill rate** (tok/s)	**Gen** (tokens)	**Decode rate** (tok/s)	**Load** (s)	**Total** (s)
Pre-upgrade allocation	2079	119.22	735	14.21	22.44	91.63
Upgraded allocation	2069	134.80	682	17.31	0.54	55.32

Table 3: Comparable live console spot checks around the infrastructure upgrade. Both runs use Gemma 4 26B with the 12-thread EMBER inference cap. They are not a controlled benchmark suite: prompt size, generated length, and model-load state differ slightly. The result is therefore used to update the interpretation, not to replace Table 2 with a full post-upgrade mean.

Deployment-bound decode ceiling, revised upward. On the preceding 16-core allocation, token generation clustered at $\approx$ 15–16 tok/s. The upgraded spot check reached 17.31 tok/s at

Call type	Prompt (tokens)	Prefill rate (tok/s)	Gen (tokens)	Decode rate (tok/s)	Load (s)	Total (s)
Chat, thinking, short (cold load)	19	88.93	295	20.28	24.06	38.8
Chat, thinking, long (warm)	29	85.41	1795	19.18	0.51	94.5
Utility, `think:false`	31	N/A	17	23.60	N/A	1.84

Table 4: Direct re-measurement of Gemma 4 26B on the current production configuration (July 6, 2026), invoking the model via its bare tag (`gemma4:26b`). The legacy `ember:latest` custom-modelfile alias referenced in earlier internal notes is not in active use. The first row includes a one-time cold model load; the second reflects steady-state warm inference.

the same 12-thread EMBER inference cap. A direct re-measurement on the current production configuration, invoking `gemma4:26b` via its bare tag and consistent with current serving practice, now converges to $\approx$ 19–20 tok/s (Table 4). We cannot fully decompose how much of this further gain traces to the CPU upgrade versus differences in serving configuration between measurement rounds, so we report the current figures as the operative reference. The qualitative conclusion holds throughout: the ceiling is deployment- and configuration-bound, not model-intrinsic.

Prompt mass still dominates total latency. Across all three measurement rounds, prefill, generated-token count, and thinking-path overhead still determine the wall-clock envelope regardless of where the decode coefficient currently sits. This continues to motivate lore gating (Section 4.4).

think:false for utility calls. The Reflect call (11 output tokens) completes in 0.57 s. The same prompt with thinking would occupy the slot for $\sim$35 s: an order-of-magnitude saving with no quality cost.

num_predict passthrough fix (v1.1.1.57). A legacy double-clamp silently made `STU_EMBER_NUM_PREDICT` entirely inoperative: `ember_num_predict()` reduced the sentinel -1 to 40; `ember_num_predict_for_model()` then unconditionally applied $\max(8192, \cdot)$, overriding every value including 40. The fix introduces an explicit passthrough:

$$f(n) = \begin{cases} 8192 & n = -1 \text{ (model default)}, \\ \max\big(40,\, \min(32768,\, n)\big) & \text{otherwise.} \end{cases} \tag{1}$$

No behavior change for the current production config ($-1 \to 8192$); any other configured value now takes effect as a genuine generation-length cap.

3.2 Model Evolution and Selection

EMBER was not designed around a single foundation model from the outset. The architecture entered continuous operation in October 2025 and evolved through several locally served Ollama model backends before converging on the current production core:

1. `phi3mini`
2. `mistral`
3. `qwen`
4. `llama 2`
5. `gemma3:12b`
6. `llama3.2-vision`
7. `gemma3:12b`

8. `gemma4:26b A4B`

The migration to `gemma4:26b A4B` occurred on April 2, 2026. This marks the introduction of Gemma 4 as the production cognitive core, not the beginning of EMBER itself. The earlier models validated PHP/Ollama integration, persistent chat state, character prompting, reflection mechanics, and vision-language browsing. The temporary return from `llama3.2-vision` to `gemma3:12b` separated vision experiments from the stable dialogue core before the final migration to the 26B MoE model.

4 Methodology and Mathematical Formalism

4.1 Asymmetric Relevance Scoring and Epistemic Consolidation

$$R(m) \;=\; \max\bigl(1,\; \min\bigl(10,\; \lfloor \alpha \cdot S_{\text{reflection}}(m) \;+\; \beta \cdot \Gamma_{\text{user}} \rfloor\bigr)\bigr) \tag{2}$$

$S_{\text{reflection}}(m) \in [0, 10]$: relevance factor assigned by the model at reflection time. Γ_{user}: configurable per-user weight (e.g. $\Gamma_{\text{Nova}} = 10$). Clamping to $[1, 10]$ prevents memory starvation and episode overweighting.

4.2 Cryptographic Deduplication via Hashing

$$\mathcal{H}(m) \;=\; \text{MD5}\Bigl(\text{Lower}\bigl(\text{RegexSub}(m,\; \texttt{"\textbackslash s+"},\; \texttt{""})\bigr)\Bigr), \tag{3}$$

$$\exists\, x \in \mathcal{R} : \text{hash}(x) = \mathcal{H}(m) \;\Longrightarrow\; \mathcal{R} \;\leftarrow\; \bigl(\mathcal{R} \setminus \{x\}\bigr) \cup \{m_{\text{updated}}\}. \tag{4}$$

4.3 Probabilistically Gated Reflection

$$\Pr\bigl[G(e) = 1\bigr] \;=\; \frac{1}{N} \cdot \mathbb{1}\bigl[\, e \notin \mathcal{V} \,\bigr], \qquad N = \texttt{STU\_EMBER\_REFLECT\_EVERY\_N}, \qquad \mathbb{E}\bigl[\text{reflection calls}\bigr] \;=\; \frac{K}{N} \cdot (1 - \rho_{\mathcal{V}}). \tag{5}$$

$\mathcal{V}$: volatile exchanges (time, date, weather, temperature). All reflection calls use `think:false`.

4.4 RAG-Lite Retrieval over Native FULLTEXT Indices (Okapi BM25)

$$\text{Score}(D, Q) \;=\; \sum_{w \in Q} \ln\!\left(1 + \frac{N - n(w) + 0.5}{n(w) + 0.5}\right) \cdot \frac{f(w, D)\,(k_1 + 1)}{f(w, D) + k_1\left(1 - b + b\,\dfrac{|D|}{\text{avgdl}}\right)}. \tag{6}$$

Boolean-mode hardening. $Q_{\text{bool}} = \bigwedge_{w \in Q}(\texttt{+}w)$ bypasses the 50% frequency threshold that suppresses high-frequency canon terms in natural-language mode.

Lore gating. $\ell(q) = \mathbb{1}[\,\texttt{useLore}(q) \wedge \text{len}(\hat{q}) > 6\,]$; only if $\ell(q) = 1$ is the BM25 block injected into u.

4.5 Prompt Mass as the Dominant Latency Driver

$$T_{\text{gen}} \;\approx\; T_{\text{prefill}}(|P|) + T_{\text{think}} + \tau \cdot n_{\text{predict}}, \quad \tau \approx 1/15.5 \text{ s/tok pre-upgrade, } 1/17.3 \text{ s/tok upgraded spot check, } 1/ \tag{7}$$

Design rule: **minimize** $|P|$**, not** $n_{\textbf{predict}}$. A real-world question with ungated lore ran to $T_{\text{total}} \approx 240 + 113 = 353$ s (two attempts); after lore gating $|P|$ dropped from $\sim$4170 to $\sim$1500–2000 characters and one attempt sufficed.

Second call in the web path: $u_2 = u \setminus (\text{LoreBlock} \cup \text{WebHint})$.

4.6 Concurrency Control: Global Advisory Lock and Per-Turn Idempotency

$\text{Lock}_{\text{global}} = \text{GET\_LOCK}(\texttt{"ember\_global\_ollama"}, 0)$. Dual-path idempotency: $\forall\,\text{id} :\ \text{render(id)} \Rightarrow \text{id} \in \mathcal{S}$; render only if $\text{id} \notin \mathcal{S}$.

4.7 Crash-Safe Presence Flags with Expiry

$\text{afk}(t) = \mathbb{1}[t < t_{\text{set}} + \Theta]$, $\Theta = 370\,\text{s} > T_{\text{cap}} = 360\,\text{s}$. $\text{visible\_afk} = \text{afk}_{\text{browse}}(t) \vee \text{afk}_{\text{console}}(t)$.

4.8 Event-Driven Vision-Language Closed-Loop Navigation

$o_t = \langle A_t,\ \text{trunc}_{1800}(\mathcal{T}_t)\rangle$. Including $\mathcal{T}_t$ was the decisive fix. Action signature loop guard: $s_t = s_{t-1} = s_{t-2} \Rightarrow$ hard abort. Resilience: result = agent finding if `done` before timeout, else top-3 snippets.

4.9 Sandboxed Python Execution and File/Video Attachments

`[PY:]` runs model-authored Python in an isolated Docker container (job-worker pattern, not a PHP subprocess), so a script cannot inherit the web process's own database credentials. The container is not given the web root. Networking runs open-bridge (full outbound internet, but no route to loopback-bound host services such as the database or inference server), a deliberate open-network-for-zero-internal-reachability trade-off, not a curated allowlist. No import/syscall blacklisting is used; isolation is enforced by the container boundary and resource caps (roughly 512 MB memory including swap, one CPU, 60 s wall clock, read-only root filesystem, dropped capabilities, unprivileged user), not by inspecting the generated code. The worker deliberately does not acquire the global inference lock (Section 4.6): the invoking request already holds it for the whole turn, so a second acquisition would deadlock.

File attachments are extracted by MIME-sniffed type (plain text direct; DOCX/XLSX/PPTX via direct ZIP/XML parsing). PDFs are handled in two stages: `pdftotext` first, and if that yields no usable text (scanned documents), a bounded number of pages is rasterized with `pdftoppm` and routed through the vision path instead, in page order rather than via OCR; the response is expected to make the resulting partial-page coverage explicit rather than imply it saw the full document. Extracted text is truncated with a head/tail split rather than tail-only truncation, then injected as an explicitly untrusted block (same framing as web/code results, Section 4.5). Video attachments are decomposed into timestamped frames via `ffmpeg` and routed through the same vision path as static images, since the served model has no dedicated video encoder but accepts multi-frame sequences; the prompt makes explicit that frames are sequential moments of one clip. Audio is not processed under any path.

4.10 SSE Window and Recency-Bounded Job Selection

`browseJobId` $= \arg\max_j(t_{\text{creation}}(j))$ where $\Delta t(j) \le 300$ s.

4.11 Ghost Cursor: Coordinate Normalization

$n_x = \text{clamp}_{[0,1]}(c_x/v_w)$, $n_y = \text{clamp}_{[0,1]}(c_y/v_h)$.

4.12 The Thinking-Bleed Phenomenon

$(\textit{think} = \texttt{true}) \wedge (\text{format} = \texttt{json}) \Rightarrow \Pr[\text{content} = \varnothing] \to 1$. Resolution: *think* = `false` for all utility calls; `think:true` is never set in any payload.

5 Autonomous Initiative Architecture (Phase 3c/3d)

Versions v1.1.1.39–v1.1.1.69 extend EMBER from a reactive to a proactive system: the entity initiates contact, follows up unanswered questions, decides autonomously when to go AFK, and reacts to self-selected news topics, all without any user trigger.

5.1 Cron-Driven Tick Endpoint and Priority Dispatch

A dedicated endpoint is called every five minutes (token-authenticated, loopback-only). Per tick at most one action is executed:

$$\text{dispatch}(t) \;=\; \begin{cases} \text{follow-up} & \text{unanswered user message} \in [T_{\min},\, 2\text{h}], \\ \text{idle} & \text{elif } \Delta t_{\text{silence}} \geq \theta_{\text{idle}}(t), \\ \text{self-AFK} & \text{elif eligibility check passes}, \\ \text{news reaction} & \text{elif } \Delta t_{\text{news}} \geq \theta_{\text{news}}(t), \\ \text{none} & \text{otherwise.} \end{cases} \tag{8}$$

All modes reuse `ember_generate_reply()`, the global `GET_LOCK` mutex, and the full `(du)`-labeled history context; no new concurrency primitives.

Follow-up mode. If the last chat line is not from the entity and has been unanswered for at least $T_{\min} = 15$ min but less than 2 h, the entity generates a reply with a system hint that it is "just back".

5.2 Idle Initiative with Deterministic Jitter

$$\theta_{\text{idle}}(t) \;=\; \theta_{\text{base}} \cdot f_{\text{self}} \;+\; J(t), \qquad J(t) = \big(\text{crc32}(\text{cid} \,\|\, t_{\text{last}}) \;\bmod\; \theta_{\text{base}}\big) \cdot \frac{1}{60}, \tag{9}$$

$f_{\text{self}} = 2$ if the entity sent the last message, else 1. Effective window: $[\theta_{\text{base}},\, 2 \cdot \theta_{\text{base}}]$ min.

Determinism is critical. A non-deterministic jitter re-sampled every 5 min could allow early firing. The fixed crc32 seed stabilizes the threshold within any silence window.

Regression: uninitialized variable (v1.1.1.49). The v1.1.1.47 patch accidentally omitted the base assignment $\theta \leftarrow \theta_{\text{base}} \cdot 60$, making $\theta = 0$ at runtime. `php -l` was silent; caught only by live observation. Lesson: always run a functional spot-check with concrete example values for arithmetic thresholds.

Prompt diversity. Four structurally distinct templates are sampled uniformly per invocation to prevent near-identical outputs from repeated identical prompts.

5.3 Self-Determined AFK

After a technical eligibility check (not already AFK; minimum inactivity $\geq T_{\text{AFK,min}} = 45$ min):

$$\{\texttt{wants\_afk} : b,\; \texttt{reason} : s\} \;=\; \pi_\theta\big(\text{"do you want to go AFK right now?"}\big). \tag{10}$$

If $b = \texttt{true}$, the existing `chat_apply_afk_state()` is called with the entity's own sender identity and s. The earlier v1.1.1.45 `mt_rand()` gate was removed in v1.1.1.46 in favor of asking unconditionally.

5.4 News Reaction with Staged Relevance Assessment

$$\theta_{\text{news}}(t) \;=\; \theta_{\text{news,base}} \;+\; J_{\text{news}}(t), \qquad \theta_{\text{news,base}} = 10\,\text{h}, \quad \text{effective: } 10\text{–}20\,\text{h}. \tag{11}$$

Three-stage pipeline: (1) topic selection via small inference call on entity's last 6 own chat lines; (2) quick retrieval via meta-search aggregator; (3) staged relevance:

$$\rho \;\in\; \{\texttt{none}, \texttt{mention}, \texttt{deep}\}. \tag{12}$$

$\rho = \texttt{none}$: timestamp updated, no action. $\rho = \texttt{mention}$: brief chat remark via `ember_generate_reply()`. $\rho = \texttt{deep}$: full Playwright browse job via `ember_browse_enqueue()`, identical to a user-triggered `[BROWSE:]` request.

5.5 Per-User Relationship Scoring (Phase 3d)

$$\delta_i \;=\; \pi_\theta^{\text{rep}}(m_i), \qquad \sigma_{u,c}^{(n)} \;=\; \sigma_{u,c}^{(n-1)} \;+\; \delta_n, \quad \delta_i \in [-10, +10]. \tag{13}$$

Stored in `stu_ember_reputation`, isolated from `ember_memories`. Tone hint:

$$h(\sigma) \;=\; \begin{cases} \text{warm hint} & \sigma \geq +30, \\ \text{cool hint} & \sigma \leq -30, \\ \varnothing & \text{otherwise.} \end{cases} \tag{14}$$

Injected into u only; separate `try/catch` from the reflection call.

6 Autonomous Tool Selection Architecture (Phase 3e)

Phase 3e (v1.1.1.62–v1.1.1.67) fundamentally restructures the initiative engine from a fixed priority chain into a deliberative orchestrator: all eligible actions are gathered per tick, and if more than one is open, the entity freely chooses among them via a single inference call.

6.1 Gate/Execute Separation and Candidate Orchestration

Every initiative action is split into two pure functions: a *gate* (cheap, no inference, checks cooldowns and state only) and an *execute* (the actual action, possibly including its own model decision such as `wants_afk` or relevance classification). The orchestrator runs as follows:

$$C \;=\; \left\{\, a \in \mathcal{A} \;\middle|\; \text{gate}_a() = \texttt{true} \,\right\}, \tag{15}$$

where $\mathcal{A} = \{\texttt{follow\_up}, \texttt{idle}, \texttt{self\_afk}, \texttt{news}, \texttt{direct\_address}\}$ is the full tool set. Given $|C|$ candidates:

$$\text{chosen} \;=\; \begin{cases} \varnothing & |C| = 0, \\ C_0 & |C| = 1 \text{ (no extra call)}, \\ \pi_\theta^{\text{choice}}(C) & |C| \geq 2. \end{cases} \tag{16}$$

$\pi_\theta^{\text{choice}}$ receives a numbered natural-language description of each open option plus "none of the above" and returns a free choice. The call uses `think:false` and the same timeout budget as the self-AFK and news reaction calls. On parse failure, the orchestrator fails safe to `none`: a silent tick is preferable to a forced action. The choice event is logged for later pattern observation.

Fallback chain (v1.1.1.66). If the chosen action's execute step internally declines (e.g. `wants_afk:false`, `relevance:none`, or a timeout), the orchestrator advances to the next untried candidate from the original gather order *without* issuing another choice call:

$$\text{for } i \in \{1, \ldots, \min(|C|, M)\}: \quad \text{result}_i = \begin{cases} \text{success} & \text{exec}(C_{\sigma(i)}) \neq \texttt{declined}, \\ \text{next} & \text{otherwise}, \end{cases} \tag{17}$$

where $M = \texttt{STU\_EMBER\_INITIATIVE\_MAX\_ATTEMPTS} \in [1, 5]$ (default 2) and σ is the original gather order. This bounds the number of inference calls per tick while ensuring slot capacity is not wasted when the preferred action is unavailable.

6.2 Tool 5: Targeted Greeting of Returning Players

The `direct_address` tool fires when a player is currently online (within the existing 90-second presence window) but has not posted a chat message for at least θ_{absence} hours:

$$\theta_{\text{absence}} \;=\; \texttt{STU\_EMBER\_DIRECT\_ADDRESS\_ABSENCE\_HOURS} \;\in\; [2,\, 72]\ \text{h} \quad (\text{default } 12\,\text{h}). \tag{18}$$

Among all eligible online players, the gate selects the one with the *largest* absence gap (deterministic, no randomness required). A per-character cooldown is persisted as a JSON map under a single `stu_kv` key (`ember_direct_address_state`), capped at 200 entries to prevent unbounded growth. The execute step generates a reply that addresses the player by name with an approximate absence duration hint.

6.3 Tool 2: Relevance-Based Long-Term Memory Retrieval

Previously, `ember_mem_fetch()` returned the top-N entries sorted globally by `relevance` and `updated_at`, independent of the current conversation topic. The replacement, `ember_mem_fetch_relevant()`, applies the existing `ember_lore_query_terms()` keyword extractor (reused verbatim) to the incoming message and scores each memory entry by the number of matching terms:

$$\text{score}(m_i,\, Q) \;=\; \bigl|\{\, w \in Q \;:\; w \in m_i \,\}\bigr| \cdot \lambda \;+\; \text{rel}(m_i), \tag{19}$$

where Q is the set of extracted query terms (up to 8, longest first), $\text{rel}(m_i)$ is the stored relevance score, and λ is a scaling factor that ensures topical term-match dominates over stored importance. The candidate pool is $\max(5N, 20)$ entries filtered by a REGEXP alternation; if no candidate matches, the function falls back to the original top-N behavior. This restores the intuition that a memory about "Nova" should surface when the user asks about Nova, even if that memory has a lower stored relevance than an unrelated high-importance entry.

6.4 Tool 3: Player State Awareness

Every prompt build now optionally includes a compact player-state block derived from the user's `stu_kv` save-state JSON blob:

$$b_{\text{state}}(u) \;=\; \langle\, \text{level},\; \text{planet},\; \Delta t_{\text{last}} \,\rangle, \tag{20}$$

where Δt_{last} is the relative time since `lastPlayedAt` (expressed as minutes / hours / days). The block is injected into u alongside the reputation hint, marked explicitly as background knowledge rather than an instruction to mention it. It is suppressed when the sender is the entity itself (initiative ticks using the entity's own identity) to avoid self-referential noise.

6.5 Ember's Own XP and Level System (v1.1.1.67)

The entity accumulates experience points using the same exponential level curve as the player client, now implemented server-side:

$$\text{XP}_{\text{needed}}(\ell) \;=\; \left\lfloor 100 \cdot 1.30^{\,\ell-1} \right\rfloor, \tag{21}$$

verified to be identical to the client-side formula for $\ell = 1, \ldots, 5$: $100, 130, 169, 219, 285$ XP. XP is granted at two hooks:

$$\Delta\text{XP} \;=\; \begin{cases} \texttt{STU\_EMBER\_SELF\_XP\_PER\_MESSAGE} \in [0, 100] & \text{on } \texttt{ember\_insert()}, \\ \texttt{STU\_EMBER\_SELF\_XP\_PER\_AFK} \in [0, 100] & \text{on successful self-AFK.} \end{cases} \tag{22}$$

`ember_insert()` is the single central insertion point for every chat message the entity posts; hooking there covers chat replies, follow-up, idle initiative, news mentions, direct addressing, and browse-result delivery without requiring per-path instrumentation. Level-up events (including carry-over XP across multiple levels in a single grant) are logged; individual grants are not, to avoid log spam.

Write safety uses the existing advisory-lock pattern: GET_LOCK("`ember_self_xp`", 3), with a 3-second wait (vs. the 0-second immediate-fail used for the global inference lock) since XP grants are infrequent and a short wait is acceptable. On lock failure the grant is silently skipped : no blocking, no overwrite.

7 Discussion: Two Paths to Persistent Agency

Savage [1] places an LLM inside a 220,000-neuron SNN with STDP; the dual-GPU requirement (RTX 5070 Ti + RTX 4060 Ti) reflects the concurrent SNN simulation workload of that substrate. EMBER pursues a different point on the same design space: rather than augmenting the LLM with a dedicated neural substrate, it supplies the LLM's attention heads with pre-filtered, high-relevance text retrieved via BM25 from a relational database. Both approaches address the same limitation, that LLMs are stateless between calls, through different memory paradigms: associative neural dynamics (Savage) vs. indexed episodic text (this work). The two are complementary rather than competing: the SNN approach yields emergent cross-domain associations through temporal co-activation, while the RAG-Lite approach offers operational simplicity, crash resilience, and CPU-only deployability. We note this only to situate our own hardware choices, not to weigh one architecture against the other.

8 Empirical Observations and Engineering Lessons

Failure mode	Root cause	Resolution
Global lore timeout (353 s)	Hardcoded 240 s curl cap ignored config	Floor 420 s / cap 480 s; config-controlled (§4.5)
Action loop produces no action	Thinking-Bleed under `format:json`	`think:false` for utility calls (§4.12)
Agent blind to page content	Observation had only accessibility-tree controls	Include $\mathcal{T}_t$ in o_t (§4.8)
Endless search re-typing	`type+submit` triggered no navigation	Action-signature loop guard + search substitution (§4.8)
CAPTCHA / 403 on search	SERPs blocked from server IP	Loopback-bound meta-search aggregator (§4.8)
Duplicate answer (slow stream)	Poll paths did not register rendered IDs	Dual-path idempotency (§4.6)
Live window never opens	Status filter excluded finished jobs (race)	Recency selection $\arg\max t_{\text{creation}}$ (§4.10)
Presence stuck on "away"	Binary flag did not survive hard abort	TTL flag $\Theta > T_{\text{cap}}$
JS fix silently inactive	Cache-buster frozen in HTML	Include cache-buster in versioning tool
Idle initiative permanently blocked	Last-sender check blocked indefinitely	Age-based threshold: $2 \times \theta_{\text{base}}$ (§5)
Idle fires every 5-min tick	Jitter patch omitted base assignment; `php -l` silent	Functional spot-check mandatory (§5)
Double idle trigger (3-min gap)	Two independent paths shared no cooldown	Unified age check against same DB row
Self-AFK: model never asked	External `mt_rand()` gated the call	Remove filter; trust `wants_afk` output (§5)
News check invisible in logs	All exit paths were silent `return false`	Log at every exit including `none` relevance (§5.4)
Choice call on single candidate	Unnecessary inference call when only one option open	Skip choice call if $\lvert C\rvert = 1$; call only at $\lvert C\rvert \geq 2$ (§6)
Memory always returns same top-N	`ember_mem_fetch()` ignored topic, sorted by stored relevance only	REGEXP-scored `ember_mem_fetch_relevant()` with top-N fallback (§6)
XP grant race on concurrent inserts	Read-modify-write on JSON blob not atomic across workers	`GET_LOCK("ember_self_xp",3)`; skip on failure (§6)

Table 5: Engineering failure modes, root causes, and architectural resolutions (v1.0–v1.2).

9 Conclusion

EMBER demonstrates that fully autonomous cognitive operation is achievable on non-accelerated CPU hardware when the architecture is designed around its constraints. The central empirical finding is that CPU-only decode throughput is substrate-bound, not model-intrinsic: the pre-upgrade allocation clustered at $\approx$ 15–16 tok/s, the upgraded 12-thread spot check reached 17.31 tok/s, and a direct re-measurement on the current production configuration converges to $\approx$19–20 tok/s. Prompt mass remains the dominant engineering lever regardless of where the decode coefficient currently sits. The autonomous initiative architecture (Phase 3c/3d/3e) scales this to proactive behavior: follow-up, idle initiative, self-determined AFK, staged news reaction, deliberative tool selection among five instruments, targeted greeting of returning players, relevance-based LTM retrieval, player-state awareness, and an entity-owned XP/level system, all without introducing new concurrency primitives. The per-user reputation scoring (Phase 3d) further personalizes the interaction. The engineering lesson catalog covers failure modes specific to CPU-only inference, single-slot contention, and cron-driven autonomous initiative.

Future work includes a single-generation optimization for the two-stage web path, longitudinal analysis of LTM and XP growth, and pattern analysis of autonomous tool selection across extended operation.

A Configuration Footprint

The configuration footprint below records a representative snapshot of the runtime control surface at the time of writing. Non-sensitive runtime values are given as illustrative examples of the architecture's operating range rather than as guaranteed, currently-live constants; they are subject to change without notice and should not be relied upon as an exact, real-time specification of the production deployment. Deployment-local identifiers, endpoint ports, and shared secrets are redacted for public release. The inline comments reflect the general semantics of the architecture, not a live configuration dump.

Listing 1: Representative configuration footprint for the EMBER CPU architecture (illustrative, subject to change).

```
<?php
// EMBER core identity and model binding.
define('STU_EMBER_ENABLED', true);                // Enables the EMBER integration path.
define('STU_EMBER_USER_ID', 0);                   // Redacted placeholder for the deployment-local EMBER service user.
define('STU_EMBER_CHARACTER_ID', '<character_id>'); // Redacted placeholder for the stable character record used by chat, memory, and initiative.
define('STU_EMBER_CHARACTER_NAME', 'Ember');      // Display name injected into runtime context.
define('STU_EMBER_MODEL', 'gemma4:26b');          // Primary Ollama model served as the cognitive core.
define('STU_EMBER_OLLAMA_URL', 'http://127.0.0.1:<ollama_port>/api/chat'); // Redacted loopback-only Ollama chat endpoint.

// CPU inference and generation controls.
define('STU_EMBER_KEEP_ALIVE', -1);               // Keep the model resident between calls; negative value means indefinite warm retention.
define('STU_EMBER_NUM_THREAD', 12);               // Dedicated inference threads on the reserved CPU partition.
define('STU_EMBER_NUM_PREDICT', -1);              // Model-default sentinel; backend passthrough/cap logic resolves the final generation budget.
define('STU_EMBER_TEMPERATURE', 0.8);             // Balanced volatility for character-stable but non-mechanical output.
define('STU_EMBER_TOP_P', 0.90);                  // Nucleus sampling threshold for output diversity control.
define('STU_EMBER_TOP_K', 40);                    // Candidate-token shortlist size for sampling.
define('STU_EMBER_REPEAT_PENALTY', 1.08);         // Mild repetition penalty to reduce loops without flattening style.
define('STU_EMBER_REPEAT_LAST_N', 64);            // Token window used by the repetition penalty.
define('STU_EMBER_SEED', -1);                     // Non-deterministic seed for natural variation between generations.
define('STU_EMBER_NUM_CTX', 8192);                // Bounded context window used to control prompt mass and memory pressure.

// Runtime cadence, transport limits, and initiative authentication.
define('STU_EMBER_IDLE_INTERVAL', 300);           // Base idle tick interval in seconds for autonomous/background checks.
define('STU_EMBER_MAX_REPLY_CHARS', 6200);        // Hard response-size guard before delivery into the chat UI.
define('STU_EMBER_TIMEOUT', 360);                 // Primary cURL execution cap for long CPU-only generations.
define('STU_EMBER_TIMEOUT_RETRY', 360);           // Retry cap for resumed or warmed follow-up calls.
define('STU_EMBER_INITIATIVE_TOKEN', '<redacted_initiative_token>'); // Redacted shared secret for cron/initiative endpoint authentication.

// Database-backed episodic long-term memory and reflection extraction.
define('STU_EMBER_MEMORY_ENABLED', true);         // Enables injection of stored episodic memory into the runtime prompt.
define('STU_EMBER_MEMORY_LIMIT', 15);             // Maximum number of memory facts injected per turn.
define('STU_EMBER_REFLECT_ENABLED', true);        // Enables post-exchange extraction of stable facts into LTM.
define('STU_EMBER_REFLECT_EVERY_N', 1);           // Reflection cadence: illustrative value; every eligible exchange in this operating mode.
define('STU_EMBER_REFLECT_TIMEOUT', 244);         // Shorter cap for utility reflection calls to protect the inference slot.
define('STU_EMBER_REFLECT_MAX_FACT_LEN', 340);    // Maximum stored fact length after reflection cleanup.

// Optional utility-model routing and moderation gates.
define('STU_EMBER_REFLECT_MODEL', '');            // Empty string falls back to STU_EMBER_MODEL for reflection.
```

```
define('STU_EMBER_IDLE_GREET_MINUTES', 60);        // Minimum absence window before a returning-user
    greeting is eligible.
define('STU_EMBER_AUTOMOD', true);                 // Global-channel basic link/vulgarity moderation gate.
```

Security and reproducibility note. This report is presented at the architectural-mathematical level. The configuration appendix is a public reproducibility footprint, not an operational specification: values are illustrative and may have since changed, and no assumption should be made that they remain in effect on the live system. Deployment-local identifiers, endpoint ports, and shared secrets are redacted. External routes, file paths, database identifiers, and account credentials are omitted.

References

[1] W. Savage, *EMBER: Autonomous Cognitive Behaviour from Learned Spiking Neural Network Dynamics in a Hybrid LLM Architecture*, arXiv:2604.12167v1 [cs.AI], April 2026.

[2] S. Robertson, H. Zaragoza, *The Probabilistic Relevance Framework: BM25 and Beyond*, Foundations and Trends in Information Retrieval, 3(4):333–389, 2009.

[3] S. Yao et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, arXiv:2210.03629, 2022.

[4] C. Packer et al., *MemGPT: Towards LLMs as Operating Systems*, arXiv:2310.08560, 2023.