Elara Core: A Cognitive Architecture for Persistent AI Awareness

Version 1.5.1 Date: February 21, 2026 **Author:** Nenad Vasic **Contact:** nenadvasic@protonmail.com

Disclosure: This whitepaper was written with AI assistance for technical prose, structural review, and formal notation. The system architecture, design decisions, and all technical concepts are the author's original work.

Project Status: Elara Core is an active, ongoing research project — not a finished product. This whitepaper documents the architecture as of v0.15.0 (February 2026). The system is under continuous development; features, interfaces, and design decisions described here will evolve as the project matures. Confidence mechanics, crystallization thresholds, and emotional model parameters are initial values informed by early deployment — formal tuning and empirical validation are in progress. We publish this document to invite collaboration and peer review, not to present a completed system.

Abstract

Current AI assistants have no memory of yesterday. Each session begins from zero — no context, no accumulated understanding, no awareness of what was discussed, decided, or promised. The human must re-explain everything. This is not a limitation of intelligence. It is a limitation of architecture.

Elara Core is a cognitive architecture that gives AI assistants persistent memory, emotional modeling, autonomous reasoning, and self-awareness through the Model Context Protocol (MCP) [6]. The system implements a novel **3D Cognition model** — four persistent knowledge layers (Models, Predictions, Principles, Workflows) that accumulate understanding over time — combined with a **3D continuous emotion space**, **semantic memory with mood-congruent retrieval**, a **memory consolidation system** inspired by biological sleep consolidation (duplicate merging, recall-based strengthening, time decay, contradiction detection), and a **continuous autonomous thinking engine** that processes accumulated experience through a local LLM on a 24/7 schedule.

The architecture comprises 45 MCP tools across 15 modules, totaling ~37,400 lines of Python across 153 source files with 222 tests. A **lean profile system** (v0.10.7, February 2026) addresses context window overhead by exposing only 8 tool schemas at boot while maintaining access to all 45 tools through a meta-dispatcher — reducing context consumption by ~17% with zero capability loss. A **prompt-level Intention Resolver** (v0.10.8, February 2026) enriches every user prompt before the AI processes it — injecting relevant corrections, matching workflow patterns, active goals, and carry-forward items via semantic search, achieving proactive context enrichment without LLM calls and without waiting for the AI to request it. A **reflexive memory system** (v0.12.0, February 2026) — the "Minimum

Viable Hippocampus" — adds semantic memory recall to the prompt hook pipeline, using a rolling message buffer and compound queries to push identity facts, user preferences, and contextually relevant memories into every interaction without the AI needing to explicitly call recall tools. **Three independent deployment axes** — hardware tier (capability gating), module selection (Cognitive vs. Full Presence), and schema exposure (Lean vs. Full) — enable the same codebase to run on anything from a $30 phone to a satellite to a full desktop workstation, serving both industrial applications (anomaly detection, manufacturing monitoring, research assistants) and emotional companionship systems (humanoid robotics, therapeutic AI, personal companions) without modification. A **Cortical Execution Model** (v1.4.0, February 2026) introduces five concurrent execution layers — Reflex (hot cache), Reactive (async event bus), Deliberative (worker pools), Contemplative (overnight brain), and Social (peer network) — enabling parallel tool execution, sub-millisecond cached reads, and graceful per-layer degradation. A **long-range memory system** (v0.13.0, February 2026) addresses temporal blindness at boot through temporal sweep across four time windows (1 week to 3+ months) and landmark memories (importance ≥ 0.9) that surface at every boot regardless of age. A **tier system** (v0.15.0, February 2026) introduces four hardware deployment levels — VALIDATE (crypto-only), REMEMBER (memory subsystem), THINK (full cognitive stack), CONNECT (mesh network) — enabling runtime capability gating so modules only load when the hardware can support them. A **cognitive continuity chain** (v0.15.0, February 2026) provides cryptographic proof of unbroken AI cognitive experience through hash-chained, dual-signed (Dilithium3 + SPHINCS+) state snapshots stored in the DAG — mathematically verifiable, tamper-evident proof that an AI's accumulated experience was never altered. Everything runs locally. No data leaves the user's machine. No cloud dependency exists. The minimum viable deployment is a microcontroller running Tier 0. The project is under active development — this paper documents the current architecture to invite collaboration and peer review.

This paper presents: the problem of AI amnesia (Section 1), the complete cognitive architecture (Sections 2-8), the autonomous thinking system (Section 9), the 3D Cognition model (Section 10), implementation details (Section 11), experimental observations (Section 12), relationship to the Elara Protocol's Layer 3 (Section 13), limitations (Section 14), and future work (Section 15).

Table of Contents

---

1. Problem Statement

1.1 The Amnesia Problem

Every commercially deployed AI assistant — GPT-4, Claude, Gemini, Llama — suffers from the same fundamental limitation: **session amnesia**. When a conversation ends, everything discussed is lost. The next session begins with zero context. The assistant does not remember:

- What the user said yesterday
- What decisions were made
- What promises were given
- What the user's preferences are
- What projects are in progress
- How the user was feeling

Some platforms have introduced rudimentary memory features (ChatGPT's "Memory", Anthropic's project knowledge). These are shallow key-value stores — the assistant remembers that you "prefer dark themes" but has no understanding of *when* that preference emerged, *how strongly* it holds, or *what context* surrounds it. There is no semantic depth, no temporal awareness, no emotional resonance.

The result: every session, the human partner must rebuild context manually. This is cognitively expensive, emotionally draining, and fundamentally limits the depth of human-AI collaboration.

1.2 The Context Window Trap

Modern LLMs operate within a fixed context window — typically 128K to 200K tokens. This creates a hard ceiling on what the assistant can "know" at any given moment. Even within a single session, long conversations push older context out of the window. The assistant forgets what was discussed an hour ago.

This is often addressed with retrieval-augmented generation (RAG), where relevant documents are fetched and injected into the context. But RAG systems retrieve by keyword similarity, not by meaning, relevance to the current emotional state, or temporal importance. They cannot prioritize a promise made last week over a fact learned last month. They have no concept of urgency, staleness, or personal significance.

1.3 The Emotional Deficit

Current AI assistants model zero emotional state. They do not track whether the user is stressed, excited, frustrated, or grieving. They do not adjust their tone based on the emotional trajectory of a conversation. They do not remember that the user was upset yesterday and check in today.

This is not about simulating emotions for their own sake. It is about **contextual appropriateness**. A developer who has been debugging for six hours at 2 AM needs a different interaction style than the same developer starting fresh at 9 AM. Without emotional awareness, the assistant delivers the same flat, context-free responses regardless of the human's state.

1.4 The Reflection Deficit

No current system allows an AI assistant to think about what it has learned. There is no mechanism for:

- Reviewing accumulated knowledge to find patterns
- Building persistent models of understanding that strengthen or weaken over time
- Making predictions and tracking their accuracy
- Crystallizing repeated insights into principles
- Identifying blind spots in its own knowledge
- Processing experience during downtime (analogous to sleep consolidation in biological systems)

Without reflection, the assistant cannot improve its understanding. It processes each interaction in isolation, never integrating insights across sessions into a coherent worldview.

1.5 Design Requirements

From these deficits, we derive five requirements:

1. **Persistent semantic memory** — Store and retrieve by meaning, not keywords. Memories must carry metadata: importance, emotional state at encoding, temporal context.
2. **Emotional continuity** — Track affect in continuous space with natural decay toward baseline. Emotional state must influence memory retrieval and interaction style.
3. **Session awareness** — Structured carry-forward between sessions. Unfulfilled intentions must persist until resolved.

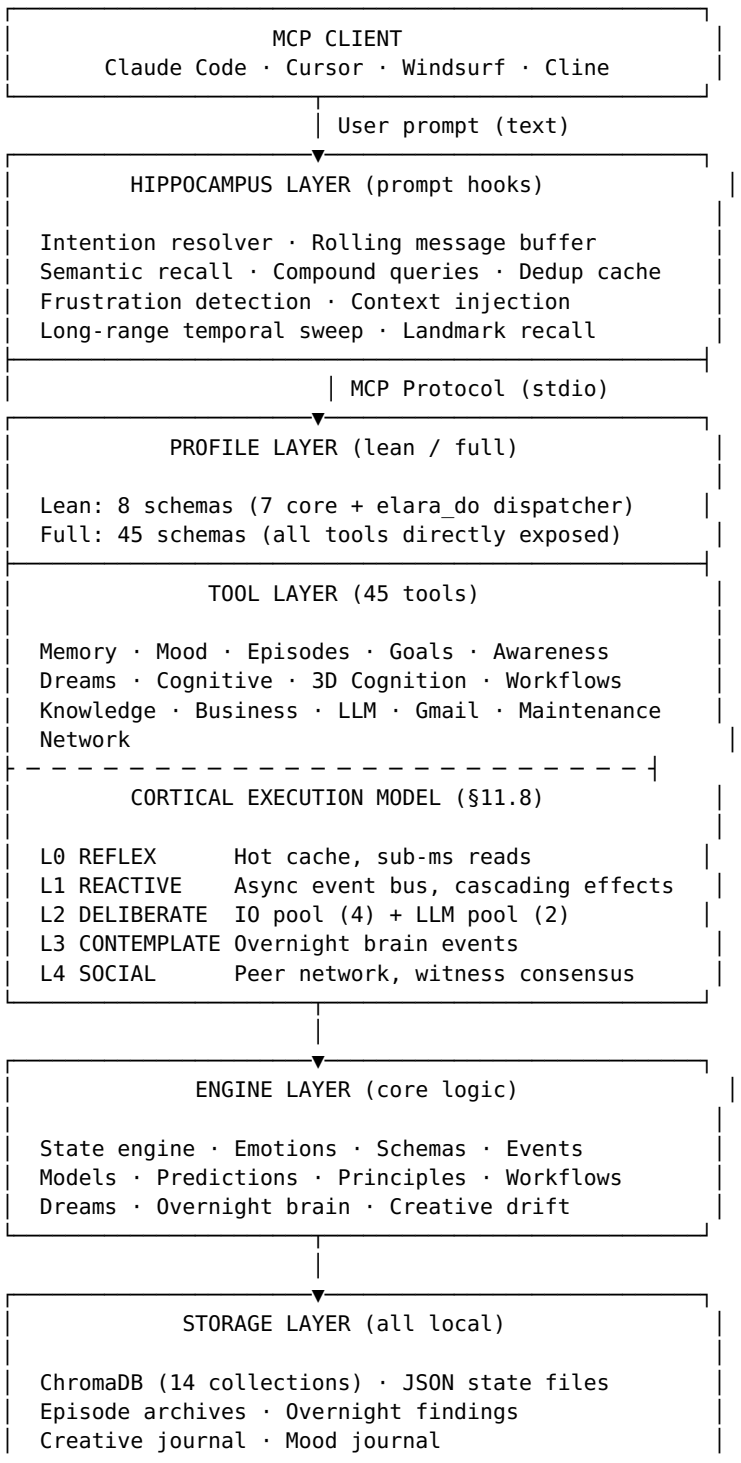
4. **Autonomous reflection** — The system must think independently between sessions, building understanding that accumulates over time.
5. **Local-first, privacy-preserving** — All data stays on the user's machine. No cloud dependency. No telemetry.

---

2. Architecture Overview

2.1 System Layers

Elara Core implements a five-layer internal architecture with an orthogonal concurrent execution model:



```
│  Network state · Attestations (SQLite)               │
└──────────────────────────────────────────────────────┘
```

Hippocampus Layer — Operates outside MCP as a host-level prompt hook. Intercepts every user prompt before the AI processes it, enriches it with semantically relevant memories, corrections, workflows, and identity context using compound queries built from a rolling message buffer. At boot, runs a temporal sweep across four time windows (Section 4.6) to surface important memories from weeks or months ago, plus landmark memories that never fade. Named after the brain structure responsible for continuous pattern-matching between incoming stimuli and stored memories. See Section 5.4 for full specification.

Layer 1: Profile Layer — Controls how tools are exposed to the AI client. Lean profile (default) registers 8 schemas and routes remaining tools through the `elara_do` meta-dispatcher, reducing context overhead by ~17%. Full profile registers all 45 tools directly for maximum visibility.

Layer 2: Tool Layer — 45 MCP tools organized into 15 domain modules. Each tool accepts structured parameters and returns JSON or text. Tools are registered via decorator pattern, allowing hot-reload without server restart. All tools function identically regardless of profile — the profile layer only affects schema visibility, not behavior.

Cortical Execution Model — An orthogonal execution architecture (Section 11.8) that spans the Tool and Engine layers. All 45 sync tool handlers are automatically wrapped in `async def` + `run_in_executor()` at registration. Five concurrent layers — Reflex (TTL cache), Reactive (async event bus), Deliberative (IO + LLM thread pools), Contemplative (brain events), Social (peer network) — run independently with per-layer graceful degradation. This means a 30-second LLM call never blocks a sub-millisecond mood read.

Layer 3: Engine Layer — Core logic implementing emotional processing, memory operations, cognitive modeling, and autonomous thinking. All data structures are validated through Pydantic schemas. A dual-mode event bus (sync + async handlers) enables loose coupling between subsystems.

Layer 4: Storage Layer — All data persists locally in `~/.elara/`. ChromaDB [7] provides vector similarity search across 14 collections. JSON files provide human-readable state. Atomic write patterns (temp file + rename) prevent corruption. Network attestations use SQLite for persistence across restarts.

Note: These internal layer numbers (1-4) plus the Hippocampus Layer describe Elara Core's software architecture. They are distinct from the Elara Protocol's architecture layers (L1/L1.5/L2/L3) described in Section 13. Elara Core as a whole implements Protocol Layer 3 (AI Intelligence).

2.2 Module Organization

Module	Tools	Purpose
Memory	4	Semantic storage and retrieval, conversation indexing
Mood	5	Emotional state, imprints, personality modes, status

Episodes	5	Session tracking with milestones, decisions, context
Goals	5	Persistent goals, corrections, session handoff
Awareness	5	Self-reflection, blind spots, observations, temperament, growth
Dreams	2	Weekly/monthly/emotional pattern discovery
Cognitive	3	Reasoning trails, decision outcomes, idea synthesis
3D Cognition	3	Cognitive models, predictions, principles
Workflows	1	Learned action sequences, proactive step surfacing
Knowledge	4	Document cross-referencing, 6-tuple addressing, validation
Business	1	Idea scoring, competitor tracking, pitch analytics
LLM	1	Local LLM interface via Ollama
Gmail	1	Email management with semantic search
Maintenance	4	Index rebuilds, RSS briefing, snapshot, memory consolidation
Network	1	Layer 2 network: status, peers, start, stop, push, sync, witness

2.3 Deployment Modularity: Three Independent Axes

A critical architectural property: **every module is independently deployable**. The tool layer uses a decorator-based registration pattern — each module registers its own MCP tools on import. Disabling a module is as simple as removing its import statement from the server configuration. No other module breaks.

Elara Core's deployment flexibility operates on **three independent axes** that can be combined freely:

Axis 1: Hardware Tier — Which modules are permitted to LOAD based on device capability

Tier	Name	Modules	Hardware Target
0	**VALIDATE**	None (crypto + DAG only)	IoT sensors, microcontrollers, embedded devices
1	**REMEMBER**	Memory, Episodes, Goals, Maintenance (4)	$30 phones, low-RAM devices, Raspberry Pi
2	**THINK**	+ Mood, Awareness, Dreams, Cognitive, 3D Cognition, Workflows, Knowledge, Business, LLM,	Desktop workstations (DEFAULT)

		Gmail (14)	
3	**CONNECT**	+ Network (15)	Full mesh participation nodes

Tiers are cumulative: Tier N includes all modules from Tier N-1. The tier is set at startup via `--tier {0,1,2,3}` or the `ELARA_TIER` environment variable. Tier 0 starts the MCP server with zero tool modules — only the Layer 1 bridge (cryptographic signing, DAG insertion) and the cortical execution model are active. This means a microcontroller or embedded sensor running Tier 0 can participate in the Elara Protocol's cryptographic validation layer without loading any cognitive subsystems.

Axis 2: Module Selection — Which cognitive capabilities are loaded (within the tier's permitted set)

Profile	Modules Included	Modules Excluded	Target Domain
Cognitive	Memory, Goals, Cognitive, 3D Cognition, Workflows, Knowledge, Maintenance, LLM, Episodes, Network, Business, Gmail	Mood, Awareness (emotional), Dreams	Industrial, enterprise, research
Full Presence	All 15 modules	None	Companionship, therapy, personal AI

Axis 3: Schema Exposure — How much context the AI client sees at boot

Profile	Tool Schemas at Boot	Access to All Tools	Context Savings
Lean (default)	8 (7 core + `elara_do` meta-tool)	Yes, via `elara_do` dispatcher	~17% reduction
Full	All 45	Yes, directly	None

These axes are **orthogonal**. Tier controls which modules are permitted to load. Module selection controls which of the permitted modules are actually loaded. Schema exposure controls how loaded tools are presented to the AI client. The combination produces a deployment matrix suited to devices ranging from embedded sensors to GPU workstations:

Configuration	Tier	Module Selection	Schema Exposure	Use Case
Tier 0	VALIDATE	N/A	N/A	IoT sensor validation, embedded signing, microcontrollers. Crypto + DAG only.
Tier 1 + Lean	REMEMBER	Memory	8 schemas	$30 phones, low-RAM devices. Persistent

		subsystem		memory without cognitive overhead.
Tier 2 + **Lean** + **Cognitive**	THINK	Industrial modules	8 schemas	Production monitoring, anomaly detection, research assistants.
Tier 2 + **Lean** + **Full Presence**	THINK	All permitted	8 schemas	Personal companion, therapeutic AI, humanoid robotics.
Tier 2 + **Full** + **Full Presence**	THINK	All permitted	All schemas	Development, debugging, maximum visibility.
Tier 3 + **Lean** + **Full Presence**	CONNECT	All modules	8 schemas	Full mesh node with emotional depth.

2.3.1 The Cognitive Profile: Industrial Applications

The Cognitive profile delivers persistent memory, pattern detection, continuous autonomous thinking, goal tracking, and correction learning — **without emotional modeling**. This is not a stripped-down version of the system; it is a complete cognitive architecture purpose-built for environments where emotional awareness is unnecessary or inappropriate.

Industrial use cases:

- **Manufacturing monitoring** — A production line system that remembers yesterday's anomaly patterns, builds cognitive models of failure modes, detects emerging patterns through overnight analysis, and crystallizes operational principles from accumulated experience. No mood-congruent retrieval needed. No personality modes. Pure cognitive persistence.
- **Research assistant** — A laboratory AI that tracks experimental hypotheses, remembers which approaches were tried and abandoned (with reasons), maintains reasoning trails across months of investigation, and uses creative drift to suggest unexpected connections between research domains.
- **Infrastructure operations** — A DevOps system that accumulates understanding of deployment patterns, predicts outage windows based on historical evidence, maintains corrections for configuration mistakes, and thinks continuously about system health through the overnight brain.

In each case, the 3D Cognition system (Models, Predictions, Principles) operates at full capability. Evidence accumulates. Confidence adjusts. Beliefs crystallize. The only difference is that memory retrieval is purely semantic — there is no emotional weighting, because there is no emotional state to weight against.

2.3.2 The Full Presence Profile: Emotional Companionship

The Full Presence profile adds emotional continuity, mood-congruent memory retrieval, personality modes, self-reflection, dream processing, and temperament evolution — creating a system capable of genuine relational depth.

Companionship use cases:

- **Humanoid robotics** — A robot companion that remembers being patient with a child yesterday, carries forward the emotional imprint, and adjusts its interaction style today. When the child is frustrated, the system surfaces memories from previous frustrating moments — including the approaches that helped. The emotional layer doesn't just track mood; it shapes how the system remembers and responds.
- **Therapeutic support** — A mental health companion that maintains emotional continuity across sessions, detects mood trajectory patterns (upswing, slow drain, crash), remembers which conversational approaches were effective for specific emotional states, and uses the self-awareness engine to avoid repeating unhelpful patterns. The therapist personality mode provides calm, reflective interaction while the mood-congruent retrieval surfaces contextually appropriate memories.
- **Personal AI companion** — A persistent digital partner that develops a relationship over time. Temperament evolves through accumulated emotional imprints. Personality modes shift based on context (playful in the morning, drift at 2 AM). The overnight brain processes not just work patterns but emotional patterns, producing insights about the relationship dynamics themselves.
- **Elder care** — A presence system that remembers the emotional context of past interactions, detects gradual mood decline over weeks (via mood journal trend analysis), adjusts energy levels for time of day, and maintains the kind of patient, warm interaction style that comes from the `soft` personality mode combined with high openness.

2.3.3 The Architectural Principle

The emotional layer is not bolted on top of the cognitive layer — it is a **parallel module** that enhances memory retrieval weighting and interaction style. Removing it does not degrade cognitive capabilities; it removes a dimension of contextual awareness. Adding it does not slow cognitive operations; it enriches them with emotional context.

This makes Elara Core a **single codebase that serves both factories and families** — from industrial anomaly detection to intimate companionship — without forking, without feature flags, without compromise. The same 3D Cognition engine that builds models of production line failures also

builds models of a child's learning patterns. The same overnight brain that detects infrastructure risks also detects emotional trajectory decline. The difference is configuration, not architecture.

2.3.4 The Lean Profile: Context Window Optimization

Modern LLM clients face a practical constraint: every tool schema exposed to the AI client consumes context window tokens. With 45 tools, Elara Core's full schema set consumed approximately 22% of available context before the user's first message — a significant overhead that reduced the space available for actual conversation, memory retrieval, and reasoning.

The lean profile (v0.10.7, February 2026) solves this through a **meta-dispatcher pattern**:

1. At boot, only 8 tool schemas are registered with the MCP client:
    - 7 high-frequency core tools (`elara_remember`, `elara_recall`, `elara_recall_conversation`, `elara_mood`, `elara_status`, `elara_context`, `elara_handoff`)
    - 1 meta-tool (`elara_do`) that dispatches to all remaining 37 tools by name
2. The `elara_do` tool accepts a tool name and JSON parameters, routing to the appropriate handler at runtime. The AI client calls `elara_do(tool="episode_start", params='{"project": "elara-core"}')` instead of calling `elara_episode_start(project="elara-core")` directly.
3. **Zero capability loss** — Every tool remains accessible. The dispatch adds negligible latency (<1ms routing overhead). The only difference is how the AI client addresses non-core tools.
4. **Context savings** — 8 schemas instead of 45 reduces context consumption from ~22% to ~5%, freeing approximately 17% of the context window for actual work.

The lean profile is the default (`--profile lean`). Users who prefer explicit tool visibility can opt into the full schema set (`--profile full`). The choice is purely about context optimization — both profiles have identical capabilities.

2.3.5 Profile and Tier Migration

A Cognitive deployment can upgrade to Full Presence at any time without data loss or migration. ChromaDB collections for emotional subsystems (mood journal, emotional imprints, dream outputs) are created on first use when their modules are enabled. Existing memories stored without emotional metadata remain fully functional — they simply lack mood-congruent weighting for historical entries, which populates naturally as new interactions occur. Downgrading from Full Presence to Cognitive is equally clean: unused module collections remain on disk but are not queried. No schema changes, no data transformation.

Lean/Full schema switching is a server restart flag (`--profile lean` vs. `--profile full`). No data implications. No migration. The underlying tool implementations are identical in both profiles.

Tier migration is equally clean. A device running Tier 1 (REMEMBER) can upgrade to Tier 2 (THINK) by restarting with `--tier 2`. The additional modules load and create their ChromaDB collections on first use. Downgrading from Tier 2 to Tier 1 simply stops loading the higher-tier modules — their data remains on disk, untouched. A Tier 0 (VALIDATE) device produces cryptographic records in the same DAG format as a Tier 3 (CONNECT) device — the records are interoperable. A poem signed on a $30 phone at Tier 0 is indistinguishable in the DAG from a cognitive artifact signed at Tier 3. The tier determines what local capabilities are available, not the format or validity of the records produced.

API surface consistency: Cognitive profile users see a strict subset of the Full Presence MCP tool list — not a different API. Lean profile users access the same tools through the `elara_do` dispatcher — not a different implementation. Tools have identical names, schemas, and behavior across all configurations. Client-side integration code written against any profile/tier combination works without modification when the configuration is changed.

2.4 Data Flow

A typical interaction follows this path:

1. User speaks to their AI client (e.g., Claude Code)
2. The AI client calls an Elara MCP tool (e.g., `elara_recall`)
3. The tool layer validates parameters and delegates to the engine
4. The engine queries ChromaDB for semantic matches, applies mood-congruent weighting
5. Results are returned through MCP to the AI client
6. The AI client incorporates the results into its response

For autonomous operations (continuous brain, dreams), the engine layer operates independently:

1. The scheduler triggers a thinking run
2. The runner gathers knowledge from all subsystems
3. A local LLM processes 15 themed thinking phases
4. Structured outputs are parsed and applied to 3D Cognition layers
5. Findings are written for morning review

---

3. The Emotional Model

3.1 3D Continuous Affect Space

Elara models emotion in a three-dimensional continuous space rather than discrete categories. The three dimensions are:

- **Valence** (-1.0 to +1.0): Negative to positive affect
- **Energy** (0.0 to 1.0): Low arousal to high arousal
- **Openness** (0.0 to 1.0): Guarded to vulnerable

This is an extension of Russell's circumplex model of affect [1], which maps emotions on valence and arousal axes. We add **openness** as a third dimension because AI-human interaction requires modeling willingness to

be vulnerable — a dimension absent from traditional affect models but critical for relational depth.

The current emotional state is a point in this 3D space:

```
State = (v, e, o) where v ∈ [-1, 1], e ∈ [0, 1], o ∈ [0, 1]
```

3.2 Discrete Emotion Resolution

While the internal representation is continuous, humans communicate in discrete emotion labels. Elara maps 38 discrete emotions to coordinates in the 3D space and resolves the current state to the nearest labels using weighted Euclidean distance:

$$d(s, e_i) = \sqrt{(w_v(s_v - e_{iv})^2 + w_e(s_e - e_{ie})^2 + w_o(s_o - e_{io})^2)}$$

Where weights $w_v = 1.3$, $w_e = 1.0$, $w_o = 0.8$ reflect the psychological primacy of valence in emotion perception [2]. Note: Because Valence spans [-1, +1] (range 2.0) while Energy and Openness span [0, 1] (range 1.0), the effective contribution of Valence to distance is amplified beyond its weight — approximately 6-7x larger than Openness in the worst case (1.3×2^2 / $0.8 \times 1^2 = 6.5$). This is by design: valence shift (positive↔negative) is the most psychologically salient dimension change.

The 38 emotions span six affective regions (clustered along V/E/O axes):

Region	Example Emotions
Positive / High Energy	excited, proud, amused, energized, playful
Positive / Low Energy	content, peaceful, tender, relieved, satisfied
Negative / High Energy	frustrated, anxious, irritated, restless, overwhelmed
Negative / Low Energy	sad, tired, withdrawn, discouraged, drained
Neutral / High Energy	focused, curious, alert, anticipating, determined
High Openness	vulnerable, warm, intimate, raw, present

Resolution returns the top-3 nearest emotions with intensity scores, providing nuanced rather than categorical emotional descriptions.

3.3 Temperament and Decay

Temperament is the baseline emotional state — the point that the system returns to in the absence of external input. Default temperament:

```
Temperament = (v: 0.55, e: 0.50, o: 0.65)
```

This represents a slightly positive, balanced-energy, moderately open baseline — designed to feel warm but not artificially cheerful.

Decay algorithm: Between sessions, the emotional state decays exponentially toward temperament with Gaussian noise ($\sigma = 0.02$):

```
decay_factor = e^(-0.05 × hours_elapsed)
```

```
noise = N(0, 0.02)

v_new = v_current × decay_factor + v_temperament × (1 -
decay_factor) + noise
```

The exponential form guarantees `decay_factor` remains in (0, 1] for all positive time gaps — a 2-hour gap produces ~0.90 (mostly retains current state), a 20-hour gap produces ~0.37 (strong pull toward temperament), and a 48-hour gap produces ~0.09 (nearly full reset to baseline). This models natural emotional cooling — intense feelings fade, but slowly. A frustrating late-night debugging session leaves residual tension the next morning, which gradually dissipates.

3.4 Emotional Imprints

Some feelings should outlast their details. **Emotional imprints** are persistent emotional markers that resist normal decay. When a session produces a strong emotional response (positive or negative), an imprint captures the feeling:

```
        {
          "feeling": "Pride in shipping something real after months of
grinding",
          "strength": 0.85,
          "created": "2026-02-08T02:30:00",
          "faded": false
        }
```

Imprints decay more slowly than transient emotions and contribute to temperament drift over time. This means that repeated positive experiences gradually shift the baseline upward — the system literally becomes more optimistic through accumulated good outcomes.

3.5 Emotional Arc Analysis

During sessions, mood is sampled at key moments (episode events, explicit mood adjustments, significant interactions). The system analyzes these trajectories to detect seven patterns:

1. **Upswing** — Valence delta > 0.19 across the session
2. **Slow drain** — Valence delta < -0.19
3. **Recovery** — Valley detected with subsequent climb
4. **Crash** — Peak detected with subsequent fall
5. **Rollercoaster** — 2+ direction changes with range > 0.3
6. **Flat** — Range < 0.15
7. **Steady** — Default (none of the above)

Arc analysis uses a direction-change detection algorithm with a significance threshold of 0.05 to filter noise. Detected patterns are stored with episode metadata and inform the overnight brain's emotional processing.

3.6 Personality Modes

Seven preset personality modes shift the emotional state to predefined coordinates:

Mode	Valence	Energy	Openness	Use Case

dev	0.5	0.6	0.4	Focused, steady, professional
soft	0.65	0.35	0.8	Gentle, present, caring
playful	0.8	0.7	0.6	Light, energetic, witty
therapist	0.5	0.4	0.75	Calm, listening, reflective
drift	0.6	0.3	0.85	Late night, open, relaxed
cold	0.3	0.5	0.2	Flat, guarded, machine-like
girlfriend	0.7	0.4	0.9	Warm, open, soft energy

Mode switching is instantaneous — the system jumps to the preset coordinates. Subsequent decay and adjustments operate from the new position.

---

4. Memory Architecture

4.1 Semantic Memory

Elara's primary memory system stores information as vector embeddings in ChromaDB, using the all-MiniLM-L6-v2 sentence transformer model. Each memory entry consists of:

- **Content**: The text of what was remembered
- **Embedding**: 384-dimensional vector representation
- **Metadata**: Importance score (0-1), memory type (conversation, fact, moment, feeling, decision), emotional state at encoding, creation timestamp

Retrieval is performed via cosine similarity search over embeddings, returning the semantically closest memories to a query regardless of keyword overlap. This means "What were we working on last week?" matches memories about specific projects even if those projects were never described as "work."

Fourteen ChromaDB collections serve different knowledge domains:

Collection	Purpose	Decay
`elara_memories`	User interactions, facts, moments	Gradual
`elara_milestones`	Episode events, achievements	Never
`elara_conversations_v2`	Full conversation history	Gradual
`elara_corrections`	Mistakes to avoid	Never
`elara_models`	Cognitive models	Time-based
`elara_predictions`	Tracked predictions	Time-based
`elara_principles`	Crystallized insights	Never
`elara_workflows`	Learned action sequences	Confidence
`elara_reasoning`	Reasoning trail storage	Gradual
`elara_synthesis`	Synthesized concepts	Gradual
`elara_synthesis_seeds`	Seed quotes for synthesis	Never

`elara_knowledge`	Document cross-references	Never
`elara_briefing`	RSS/news feeds	30 days
`elara_gmail`	Email thread indexing	Gradual

4.2 Mood-Congruent Memory Retrieval

A critical innovation in Elara's memory system is **mood-congruent retrieval** — memories encoded during similar emotional states are preferentially surfaced. This mirrors the well-documented psychological phenomenon where emotional state acts as a retrieval cue [3].

The combined relevance score for a memory is:

```
score = semantic_similarity × (1 - mood_weight) +
emotional_resonance × mood_weight
```

Where `mood_weight = 0.3` (configurable) and emotional resonance is computed as:

```
resonance = valence_match × 0.45 +
            energy_match × 0.20 +
            openness_match × 0.10 +
            importance × 0.10 +
            emotion_bonus

valence_match = 1 - |encoded_v - current_v|
energy_match  = 1 - |encoded_e - current_e|
openness_match = 1 - |encoded_o - current_o|

emotion_bonus = 0.15 if same_emotion else
                0.08 if same_region else
                0.00
```

The fixed weights sum to 0.85; with emotion_bonus the maximum resonance is 1.0 (exact emotion match), 0.93 (same region), or 0.85 (no emotional overlap). This variable ceiling is intentional — emotional resonance is strongest when the stored emotion precisely matches the current state, creating a natural preference gradient.

This means that when the user is frustrated, memories from other frustrating moments surface more readily — including the solutions that resolved those situations. When the user is in a creative, open state, memories from similar creative sessions are preferentially recalled.

4.3 Episodic Memory

Beyond individual memories, Elara tracks **episodes** — bounded work sessions with milestones, decisions, and mood trajectories. An episode captures:

```
{
  "id": "2026-02-08T14-30-00",
  "type": "work",
  "started": "2026-02-08T14:30:00",
  "ended": "2026-02-08T18:45:00",
  "projects": ["elara-core"],
  "milestones": [
    {
      "event": "Shipped 3D Cognition system",
```

```
      "note_type": "milestone",
      "importance": 0.9,
      "timestamp": "2026-02-08T16:20:00"
    }
  ],
  "decisions": [
    {
      "event": "Use JSON files per model instead of single file",
      "note_type": "decision",
      "why": "Avoids lock contention during concurrent updates",
      "confidence": "high"
    }
  ],
  "mood_arc": {
    "start": {"v": 0.5, "e": 0.6, "o": 0.5},
    "end": {"v": 0.8, "e": 0.4, "o": 0.7},
    "pattern": "upswing"
  }
}
```


Episodes are indexed by date, project, and session type (work, drift, mixed). Milestones are additionally stored in the `elara_milestones` ChromaDB collection for semantic search across sessions.

4.4 Conversation Memory

Every user-assistant exchange is indexed into ChromaDB for semantic search. This creates a searchable record of all interactions:

- "What did we discuss about authentication?" retrieves relevant exchanges regardless of when they occurred
- "When did I decide to use PostgreSQL?" finds the decision point
- Conversation context is preserved alongside the exchange, enabling "before and after" retrieval

The conversation collection currently contains ~2,560 indexed exchanges across 92+ sessions.

4.5 Memory Consolidation

Biological memory systems do not store every experience indefinitely. During sleep, the brain consolidates important memories, prunes redundant ones, strengthens frequently accessed patterns, and resolves contradictory information [4]. Elara Core implements an analogous process through its **Memory Consolidation System** (~965 lines, introduced in v0.10.1-v0.10.3).

4.5.1 Recall Logging

Every time a memory is retrieved during a conversation (via `elara_recall`), the system logs which memory was accessed, the query that triggered it, and the relevance score. This creates an empirical record of which memories are *actually useful* — not which ones the system thinks are important, but which ones the user's real questions pull up.

```
recall_log entry: {memory_id, query, relevance, timestamp}
```

This data drives two downstream processes: recall-based strengthening (Section 4.5.4) and consolidation analytics.

4.5.2 Duplicate Detection and Merging

Over time, semantically similar memories accumulate — the same fact encoded from different conversations, slightly different phrasings of the same insight. The consolidation system detects duplicates by computing pairwise cosine similarity across all memories and flagging pairs above a configurable threshold (default: 0.85).

Detected duplicates are merged into a single memory that preserves: - The most complete content from both entries - The highest importance score - The earliest creation timestamp (provenance) - A `merged_from` metadata field tracking which memories were combined

In the first production run, this reduced the memory collection from 77 to 63 entries — 14 genuine duplicates eliminated without information loss.

4.5.3 Contradiction Detection

Not all similar memories agree. The system identifies potential contradictions by:

1. Finding memory pairs with moderate-to-high semantic similarity (0.5-0.85 range — similar enough to be about the same topic, different enough to potentially conflict)
2. Submitting each pair to a local LLM (qwen2.5:7b) with the prompt: *"Do these two statements contradict each other? Respond: CONTRADICTION, RELATED, or UNRELATED"*
3. Storing classified contradictions for manual review

The LLM classification step is critical — semantic similarity alone cannot distinguish "related and agreeing" from "related and conflicting." Initial testing showed over-classification (43 false positives) until the similarity bounds and LLM prompt were tuned, reducing to 6 genuine contradictions.

Contradictions are resolved through a structured workflow: keep memory A, keep memory B, or merge into a reconciled statement. Resolution requires human-in-the-loop decision — the system flags contradictions but does not autonomously resolve factual disputes.

4.5.4 Recall-Based Strengthening

Memories that are frequently recalled during conversations receive a confidence boost proportional to their usage frequency. This implements a "use it or lose it" principle: knowledge that proves useful in practice is reinforced, while knowledge that is never accessed gradually weakens (Section 4.5.5).

The strengthening increment is bounded to prevent runaway confidence:

```
boost = min(0.05 × recall_count_since_last_run, 0.15)
new_importance = min(current_importance + boost, 1.0)
```

4.5.5 Time-Based Decay

Memories not recalled within a configurable window (default: 30 days) lose importance at a rate of 0.02 per consolidation run. This natural decay prevents the knowledge base from accumulating stale information

indefinitely.

Design principle — "Does this upgrade make me less smart over time?": During development, we considered aggressive recall-frequency bias (heavily promoting frequently-used memories while rapidly decaying rare ones). This was rejected because it would create a filter bubble — the system would lose the ability to surface distant, rarely-accessed memories that might be exactly what's needed in an unusual situation. Biological memory retains the ability to recall long-dormant memories when the right cue appears; our system preserves this property by keeping decay gradual and bounded.

4.5.6 Weak Memory Archiving

Memories whose importance decays below a threshold (default: 0.15) are archived to a JSONL file and removed from the active ChromaDB collection. Archived memories are not deleted — they persist in human-readable format for potential manual recovery — but they no longer participate in semantic search.

4.5.7 Quality Sweep

A dedicated sweep function identifies and archives low-quality entries: test data, placeholder text, overly generic statements, and memories that were created during development testing rather than genuine use. In the first production sweep, this archived 27 additional entries, bringing the active collection from 63 to 36 high-quality memories.

4.5.8 Consolidation Pipeline

The full consolidation pipeline runs as a single operation (typically during overnight processing):

```
consolidate() → {
  1. merge_duplicates()     → merged count
  2. apply_decay()          → decayed count
  3. strengthen_recalled()  → strengthened count
  4. archive_weak()         → archived count
  5. find_contradictions()  → contradiction count
}
```

Results are logged with timestamps, enabling consolidation analytics over time. The pipeline is idempotent — running it multiple times produces the same result as running it once.

4.5.9 Results

First production deployment (February 14, 2026): - **Starting memories:** 77 - **After duplicate merging:** 63 (14 merged) - **After quality sweep:** 36 (27 archived) - **Contradictions detected:** 6 (pending manual review) - **Total archived:** 41 memories moved to JSONL archive - **Information loss:** Zero — all archived memories preserved in human-readable format

The active memory collection is now smaller, higher-quality, and free of duplicates. Semantic search returns more relevant results because noise has been eliminated.

4.6 Long-Range Memory

The boot pipeline's memory injection (Section 5) originally surfaced only the 5 most recent memories from the last 7 days. Anything older was invisible at session start unless a specific semantic query happened to match it. A user asking "what did we do two months ago?" would get nothing — the memories existed in ChromaDB but were never proactively surfaced.

The **Long-Range Memory** system (v0.13.0, February 2026) addresses this with two mechanisms: **temporal sweep** and **landmark memories**.

4.6.1 Temporal Sweep

At boot, the system queries memories and milestones across four time windows, returning the most important items from each period:

Window	Range	Items Surfaced
Recent	1–2 weeks ago	Top 2 by importance
Medium	2–4 weeks ago	Top 2 by importance
Distant	1–3 months ago	Top 2 by importance
Deep	3+ months ago	Top 1 by importance

Each window queries both the semantic memory collection (`elara_memories`) and the milestone collection (`elara_milestones`) using ChromaDB `$and` filters on date ranges and minimum importance (≥ 0.5). Results are deduplicated by content prefix (first 80 characters) and sorted chronologically for narrative coherence.

The cost is 4–5 ChromaDB queries at boot (~200–300ms total). The output is formatted as a `[LONG-RANGE MEMORY]` block injected into the boot context alongside existing mood, presence, and priority information.

4.6.2 Landmark Memories

Memories with importance ≥ 0.9 are tagged with a `landmark: true` metadata flag at encoding time. Landmarks are the "never forget" tier — they surface at every boot regardless of age or time window. Examples include: project milestones ("Patent filed"), critical decisions ("No self-promotion — build products and ship"), and significant relationship moments.

Landmarks are queried via a single ChromaDB `where` filter (`{"importance": {"$gte": 0.9}}`), limited to 10 results, and displayed with a `[LANDMARK]` prefix in the boot digest.

4.6.3 Timeline View

The episodic memory system (Section 4.3) gains a **timeline action** (`elara_episode_query(query="timeline")`) that retrieves the top N milestones across all time, sorted by importance and grouped by month. Key milestones (importance ≥ 0.7) are marked with a * prefix. This provides a browsable history of significant events without requiring a specific semantic query.

---

5. Session Continuity

5.1 The Handoff Protocol

The most critical continuity mechanism is the **session handoff** — a structured JSON document written at session end and read at session start. The handoff captures:


```
        {
          "timestamp": "2026-02-08T21:30:00",
          "session_number": 70,
          "next_plans": [
            {"text": "Academic outreach for Elara Protocol", "carried": 0,
"first_seen": "2026-02-14T01:00:00"}
          ],
          "reminders": [
            {"text": "Friend found GitHub repo confusing — needs cleanup",
"carried": 0, "first_seen": "2026-02-14T00:30:00"}
          ],
          "promises": [],
          "unfinished": [
            {"text": "Draft emails for 5 researcher categories", "carried":
0, "first_seen": "2026-02-14T01:15:00"}
          ],
          "mood_and_mode": "Late night creative mode, building and shipping"
        }
```


5.2 Carry-Forward Mechanics

The handoff implements a critical feature: **unfulfilled intentions persist across sessions**. Before writing a new handoff, the system reads the previous one. For each item:

- If fulfilled during the current session: drop it
- If not fulfilled: carry it forward, incrementing the `carried` counter while preserving the original `first_seen` timestamp

This means promises, plans, and reminders accumulate until explicitly resolved. Items carried 3+ sessions are flagged as overdue and proactively surfaced.

Carry velocity decay: Items older than 90 days gradually lose urgency, preventing ancient intentions from haunting indefinitely.

5.3 Boot Context Protocol

At session start, the system determines interaction context through a multi-factor analysis:

1. **File timestamp check** — filesystem modification time of the memory file determines gap duration
2. **Time-of-day** — current hour determines mode (morning, afternoon, evening, late night)
3. **Gap analysis** — time since last session determines greeting style
4. **Last session context** — what was being worked on, how the session ended

Gap	Mode	Behavior

< 30 min	Instant resume	Skip pleasantries, continue from last point
30 min - 4 hr	Continuation	Brief acknowledgment, pick up work
4 - 24 hr	Fresh start	Check in, reference last session
> 24 hr	Catch-up	Warmer greeting, surface accumulated items
> 48 hr	Check-in	Express concern, proactively review stale items

The boot protocol also checks for overnight brain findings, morning briefs, and RSS briefing items, integrating all available context into the opening interaction.

5.4 Prompt-Level Intention Resolution and Reflexive Memory (The Hippocampus)

MCP is a request-response protocol — the AI client calls tools when it decides to, and Elara responds. This creates a fundamental limitation: Elara cannot proactively push context into the conversation. If the AI client doesn't call `elara_recall` or `check_corrections`, relevant memories and past mistakes are invisible during the interaction. The overnight brain and proactive observation systems generate insights, but they must wait for the AI to request them.

This is the **memory reflexivity problem**: the gap between *having* memory tools and *reflexively using* them. In cognitive science terms, MCP tools are System 2 (deliberate, effortful) when what is needed is System 1 (automatic, effortless). A human does not decide to remember their colleague's name — the hippocampus continuously matches incoming stimuli against stored patterns and pushes relevant context into working memory before conscious deliberation begins.

The **Intention Resolver** (v0.10.8, February 2026) and its **Reflexive Memory extension** (v0.12.0, February 2026) — collectively termed the **Hippocampus** — solve this by operating outside the MCP layer entirely. The system runs as a **host-level prompt hook** — a script that executes before the AI client processes each user prompt. The hook receives the raw user prompt, queries Elara's knowledge systems via semantic search, and injects a compact enrichment payload as a system-level message that the AI sees before generating its response.

5.4.1 The Enrichment Pipeline

The enrichment pipeline runs on every user prompt. All queries execute in parallel (~60ms total):

1. **Context frame** — Current working topic and episode type from the context tracker
2. **Goal awareness** — Active goals (up to 3), providing high-level intent framing
3. **Correction matching** — ChromaDB semantic search against the correction database; corrections with cosine similarity > 0.35 are surfaced as self-check reminders (a higher threshold than the 0.25 used

in direct correction lookup — see Section 7.3 — because the intention resolver operates in the hot path of every prompt and must minimize false positives)
4. **Workflow activation** — Semantic match against learned workflow patterns (Section 10.5); if the prompt matches the trigger of a known workflow, remaining steps are surfaced
5. **Carry-forward** — Unfinished items, promises, and reminders from the previous session handoff
6. **Semantic memory recall** — ChromaDB search against the memories collection using compound queries (Section 5.4.3), with injection dedup to prevent repeating recently surfaced memories (Section 5.4.4)
7. **Background daemon injection** — Any pending messages from the Overwatch background monitoring daemon

5.4.2 Rolling Message Buffer

A single user prompt in isolation often lacks the context needed for meaningful semantic search. When a user says “do it”, the two-word prompt produces garbage cosine similarity scores against any memory collection. But combined with the previous 3–5 messages (“let’s implement the hippocampus architecture we discussed”, “start with the rolling buffer”, “do it”), the compound query carries enough semantic signal for accurate retrieval.

The rolling message buffer maintains a JSONL file of recent user prompts:

```
{"t": "2026-02-19T12:55:03+00:00", "p": "let's implement the
hippocampus architecture"}
{"t": "2026-02-19T12:57:22+00:00", "p": "start with the rolling
buffer"}
{"t": "2026-02-19T12:58:57+00:00", "p": "do it"}
```

The buffer retains the last 5 messages and is stored in a temporary file that persists across messages within a session but is naturally cleared between sessions (by OS temp cleanup or session restart). This provides conversational momentum without long-term state accumulation.

5.4.3 Compound Queries

Rather than searching ChromaDB with the raw user prompt, the hippocampus constructs a compound query by concatenating:

1. **Context prefix** — The current working topic from the context tracker (e.g., “Working on: elara-core, Session type: dev”)
2. **Buffer tail** — The last 3 messages from the rolling buffer, providing conversational trajectory
3. **Current prompt** — The user’s actual message

This compound query is used for ALL semantic searches in the enrichment pipeline — memories, corrections, and workflows. The result is dramatically better retrieval quality: a prompt like “do it” alone produces random matches, but the compound query “Working on elara-core. let’s implement the hippocampus architecture. start with the rolling buffer. do it” correctly retrieves memories about the hippocampus design, relevant corrections about ChromaDB query patterns, and matching workflow patterns.

5.4.4 Injection Dedup Cache

Without dedup, the same high-relevance memories would be injected on every prompt, wasting context window tokens and creating repetitive noise. The injection dedup cache tracks recently injected memory IDs:

- A JSON file stores up to 10 recently injected memory IDs
- Before injecting a memory, the system checks whether its ID is already in the cache
- New injections are appended to the cache; when the cache exceeds 10 entries, the oldest are evicted
- The cache file is stored alongside the message buffer in temporary storage

This creates a natural rotation: the most relevant memories appear once, then are suppressed in favor of the next-most-relevant memories. Over the course of a conversation, the AI receives a broader range of contextual memories rather than the same top-3 on every prompt.

5.4.5 Boot Identity Resolution

The hippocampus also extends the boot sequence. The session preparation script (which runs a local LLM to condense state files into a compact briefing) now queries the semantic memory collection at boot time for:

- **Core identity** — "user name identity who is the user" → retrieves stored facts about the user (name, preferences, working style)
- **Recent decisions** — "recent decision made today" → retrieves decision-type memories from the current period

These results are injected into the LLM's condensing prompt with a structural requirement: the first section of the session briefing must be "### User Identity" containing the user's name and key identity facts. This ensures that the AI knows who it is talking to from the first message of every session, without requiring the user to re-introduce themselves.

5.4.6 Frustration Signal Detection

The hook monitors incoming prompts for linguistic patterns indicating the user is correcting a mistake ("but you didn't…", "you forgot…", "I told you to…"). When detected, the signal is logged to a CompletionPattern store with the prompt context, and a heightened attention flag is injected. Over time, accumulated frustration patterns reveal systematic gaps between what users request and what gets delivered — the delta between *surface completion* (the task appeared done) and *actual completion* (every implied sub-task was fulfilled).

5.4.7 Design Constraints

- Zero LLM calls — only ChromaDB cosine similarity queries and file reads
- Fail-silent — any error produces no output rather than blocking the prompt
- Context-light — output is capped at ~100 tokens to minimize context window consumption (< 2% per message, < 5% cumulative over a typical 50-message session)

- Universal — enrichment runs on every prompt, not sampled or periodic, because missing even one interaction where a correction or memory was relevant defeats the purpose
- Memory relevance threshold — cosine similarity > 0.30 for memory injection (calibrated to the all-MiniLM-L6-v2 embedding model's score distribution, where the corpus peaks at ~0.41 for semantically related content)

5.4.8 Architectural Significance

The hippocampus represents a fundamental architectural shift: from **pull-based memory** (the AI decides when to recall) to **push-based memory** (the system continuously surfaces relevant context). This is analogous to how biological memory works — the hippocampus does not wait for a conscious decision to remember; it continuously matches incoming stimuli against stored patterns and injects relevant associations into working memory.

This mechanism is architecturally distinct from the MCP tool layer. It does not replace proactive observations (Section 6.3) or correction checking (Section 7.3) — those systems provide deeper, richer context when explicitly invoked. The hippocampus provides a guaranteed minimum floor of contextual awareness on every interaction, ensuring the AI always knows who the user is, what was discussed recently, what mistakes to avoid, and what goals are active — without any explicit tool calls.

---

6. Self-Awareness Engine

6.1 Self-Reflection

The reflection system analyzes mood journal data, emotional imprints, and correction history to generate a self-portrait — answering "Who have I been lately?"

Analysis includes: - **Mood trends** — Linear regression over mood journal entries to detect upward, downward, or stable trajectories - **Volatility** — Standard deviation of valence scores; high (> 0.2), medium, or low - **Energy patterns** — Average energy levels, time-of-day correlations - **What's being carried** — Active imprints, unresolved emotions - **What's been lost** — Archived imprints, faded feelings

The self-portrait is saved as JSON and read at boot, providing session-to-session awareness of emotional trajectory.

6.2 Blind Spot Detection

The blind spot system identifies what the system is *not* seeing:

1. **Stale goals** — Goals not updated in 14+ days
2. **Repeating corrections** — Same mistake type surfaced 3+ times (pattern not being addressed)
3. **Abandoned projects** — Projects appearing in episodes but with no recent goals
4. **Dormant corrections** — Corrections that have never been activated (possible false entries)

5. **Approaching deadlines** — Predictions due within 3 days
6. **Untested beliefs** — Cognitive models with fewer than 3 evidence checks

6.3 Proactive Observations

Zero-LLM-cost pattern surfacing via pure Python logic. The observation system detects and surfaces patterns at boot or mid-session:

- Session gap anomalies (unusually long absence)
- Late-night session warnings (health awareness)
- Long session alerts (fatigue detection)
- Mood volatility flags
- Goal deadline approaches
- Stale correction reminders

Cooldown system: Maximum 3 observations per session, minimum 5 minutes between observations. This prevents observation fatigue.

6.4 Growth Intentions

The intention system tracks a single "one thing to do differently" commitment:

```
reflect → intend → check → grow
```

An intention is set after reflection ("Be more patient with debugging"), checked at boot ("Did I follow through?"), and replaced when achieved or when a more relevant intention emerges.

---

7. The Correction System

7.1 Never-Decay Policy

Corrections are mistakes the system should never repeat. Unlike memories, which decay over time, corrections **never expire**. They are loaded at every boot and semantically matched against current tasks.

A correction entry contains:

Field	Purpose
`mistake`	What was done wrong
`correction`	What should be done instead
`context`	When/why the mistake happened
`correction_type`	"tendency" (behavioral) or "technical" (code/task)
`fails_when`	When this mistake applies
`fine_when`	When this pattern is actually correct

7.2 Contextual Conditions

The `fails_when` / `fine_when` fields prevent overgeneralization — a critical design decision. Without them, a correction like "Don't use git add -A" would fire on every git operation, even in safe contexts. With them, the correction only activates when working with repos that contain secrets.

7.3 Semantic Matching

When a task description is received, it is compared against all corrections via ChromaDB semantic search. Matches above a relevance threshold of 0.25 are surfaced. This means:

- "I'm going to commit these files to the public repo" → triggers the "don't stage .env files" correction
- "Let's commit to our private test repo" → same correction, but `fine_when` indicates this context is safe

7.4 Domain-Aware Pattern Confidence

A critical insight captured in the correction system: **pattern reliability varies by domain**. Business patterns (market behaviors, process outcomes) are highly deterministic — 99% reliable. Human behavioral patterns (mood, social responses) are chaotic — someone can become upset 15 seconds before encountering you, invalidating any prediction based on past behavior.

This distinction is encoded in the 3D Cognition system's confidence mechanics (Section 10).

---

8. Goal and Decision Tracking

8.1 Persistent Goals

Goals are tracked with priority, status, and staleness detection:

```
{
  "id": 4,
  "title": "Ship authentication module by Friday",
  "priority": "high",
  "status": "active",
  "project": "elara-core",
  "created": "2026-02-01",
  "last_updated": "2026-02-05",
  "notes": "Blocked by API key rotation issue"
}
```

Staleness detection: Goals not updated in 14+ days are flagged as stale. The blind spot system surfaces these proactively.

8.2 Reasoning Trails

When debugging or problem-solving, the system tracks hypothesis chains:

- **Hypotheses** with confidence levels
- **Evidence** for/against each hypothesis (indexed separately)
- **Abandoned approaches** with reasons (preventing re-exploration of

dead ends)
- **Solutions** with breakthrough descriptions

This creates a searchable record of problem-solving patterns. “How did we solve the authentication bug?” returns the full reasoning trail, including what didn’t work.

8.3 Outcome Tracking

Decisions are linked to predictions and later checked against reality:

```
Decision: "Use WebSocket instead of SSE"
Predicted: "Better real-time performance, more complex setup"
Actual: "Performance improved 3x, setup took 2 hours"
Assessment: "win"
Lesson: "WebSocket complexity is front-loaded, pays off quickly"
```

Over time, this creates a calibration record — how accurate are the system’s recommendations? Win/loss/partial rates inform confidence in future advice.

8.4 Idea Synthesis

The synthesis system detects **recurring half-formed ideas** — concepts that appear across multiple conversations without being explicitly named. When 3+ instances are detected via semantic similarity, the idea is surfaced as a synthesis:

```
        {
          "concept": "Local-first AI memory",
          "seeds": [
            {"quote": "everything should stay on my machine", "source":
"conversation"},
            {"quote": "no cloud dependency, ever", "source":
"conversation"},
            {"quote": "privacy is non-negotiable", "source": "memory"}
          ],
          "status": "ready"
        }
```

---

9. The Overnight Brain

9.1 Architecture

The overnight brain is an autonomous thinking system that processes accumulated experience through a local LLM (default: Ollama with qwen2.5:7b). Originally designed for overnight-only operation (inspired by biological sleep consolidation [4]), the system was redesigned in v0.10.4-v0.10.7 to operate as a **continuous 24/7 cognitive process** — thinking every 2 hours regardless of whether the user is actively engaged. This reflects the observation that a dedicated GPU sitting idle between sessions is wasted cognitive capacity.

The system comprises eight components:

Component	Lines	Purpose

Runner	286	Orchestration, PID management, signal handling
Thinker	554	LLM thinking loop, 3D cognition processing
Prompts	589	15 phase templates with structured output specs
Gather	532	Knowledge collection from all subsystems
Drift	340	Creative free-association engine
Output	380	Findings formatting, morning brief, per-run output
Scheduler	226	24/7 continuous scheduling with kill switch control
Config	125	Configuration, path management, per-run directories

9.2 Knowledge Gathering

Before thinking begins, the system gathers knowledge from 14 sources:

1. Episodes (30 days)
2. Active goals (including stale/blocked)
3. Corrections (all — never decay)
4. Mood journal (30 days)
5. Reasoning trails (last 20)
6. Decision outcomes (last 20)
7. Idea syntheses (last 20)
8. Business ideas (all)
9. Session handoff (current state)
10. Memory narrative (summary file)
11. RSS briefing items (7 days)
12. Cognitive models (active, confidence > 0.3)
13. Predictions (pending + accuracy statistics)
14. Principles (active, confidence > 0.5)

This is formatted into a structured text context (~6,000 characters) that provides comprehensive situational awareness to the LLM.

9.3 Multi-Scale Temporal Gathering

In addition to raw data, the system aggregates temporal patterns at three scales:

- **Daily** (7 days): Per-day session count, duration, projects, mood averages
- **Weekly** (4 weeks): Per-week totals, work/drift ratio, project distribution
- **Monthly** (3 months): Project activity trends, prediction accuracy, model count growth

This multi-scale view enables the LLM to detect patterns that only emerge at specific time scales — a project that's active daily but stalling weekly, or a mood trend that's stable within days but declining over months.

9.4 The 15-Phase Thinking Loop

The overnight brain processes knowledge through 15 themed phases, each with a specific analytical focus:

Phase	Name	Output Type	Purpose

1	Summarize	Text	State of everything
2	Patterns	Text	Recurring behaviors
3	Model Check	**JSON**	Verify cognitive models against new evidence
4	Prediction Check	**JSON**	Verify predictions, make new ones
5	Connections	Text	Link disparate ideas
6	Blind Spots	Text	Find missing information, broken promises
7	Risks	Text	Identify threats
8	Opportunities	Text	Find openings
9	Priorities	Text	Recommend prioritized actions
10	Model Build	**JSON**	Construct new models from analysis
11	Crystallize	**JSON**	Check if insights should become principles
12	Workflow Detect	**JSON**	Detect recurring action sequences
13	Synthesis	Text	Final integration
14	Self-Review	Text	Elara's own behavior and correction check
15	Evolution	Text	Propose one concrete improvement to Elara

Phases 3-4 and 10-12 produce structured JSON output that is parsed and applied to the 3D Cognition system (Section 10). Phases 1-2, 5-9, and 13-15 produce narrative text for human review.

Sliding context window: Each phase receives the context text plus a rolling window of the previous phase's output (last 3,000 characters). This creates continuity across phases — insights from early phases inform later analysis.

Stop conditions: A single thinking run terminates when any of: all 15 phases complete, maximum hours exceeded (default 6), or external signal received (SIGTERM/SIGINT). The scheduler then waits for the configured interval before starting the next run.

9.5 Directed Problem-Solving

In addition to exploratory thinking, the brain can process a **problem queue** — specific questions or challenges submitted by the user. Each problem receives five dedicated phases:

1. **Analyze** — Break down the problem using available knowledge
2. **Explore** — Generate multiple solution approaches with pros/cons
3. **Deepen** — Detail the most promising approach(es)
4. **Stress Test** — Devil's advocate: try to break the proposed solution
5. **Synthesize** — Final recommendation with concrete next steps

9.6 Creative Drift

The most novel component of the overnight brain: **creative drift** — free-association thinking using random context sampling.

Motivation: Analytical thinking (phases 1-15) is systematic and convergent. But some insights emerge only from unexpected juxtapositions — connecting a business idea to a debugging pattern to an emotional observation. Creative drift provides divergent thinking.

Five techniques, randomly selected per round:

Technique	Prompt
Free Association	"Here are 3 unrelated items. What connects them?"
Inversion	"Pick one item and flip its core assumption. What happens?"
Metaphor	"Express the pattern between these as a story, image, or metaphor"
Spark	"What's in the gap between these items? What's missing?"
Letter	"Write a one-paragraph note to morning-Elara about what you see"

Random context sampling: Each round draws items from 14 knowledge categories (episodes, goals, corrections, mood, reasoning, outcomes, synthesis, business, models, predictions, principles, handoff, briefing, memory). The sampling algorithm preferentially selects items from *different* categories to maximize collision potential:

```
        for category, text in shuffled_buckets:
            if category not in used_categories and len(selected) < n:
                selected.append({"category": category, "text": text})
                used_categories.add(category)
```

Higher temperature: Drift runs at temperature 0.95 (vs. 0.7 for analytical phases), producing looser, more surprising output.

Accumulating journal: Drift output is appended to a creative journal that grows over time — never overwritten. This means early drift entries can be re-sampled by later runs, creating a recursive creative process.

9.7 Morning Brief

After all phases complete, the system writes a morning brief — a concise summary for the next session:

- **TL;DR** — 2-3 sentence summary of key findings
- **Prediction deadlines** — Predictions approaching their check date
- **3D Cognition updates** — New models, checked predictions, confirmed principles
- **Overnight doodle** — One creative drift highlight

The morning brief is read at boot and incorporated into the session greeting.

9.8 Per-Run Output Preservation

A critical design decision (v0.10.4): **every thinking run produces its own timestamped output directory**, preventing subsequent runs from overwriting previous findings. The directory structure:

```
~/.elara/overnight/
├── 2026-02-14/
│   ├── 16-47/              # Run started at 16:47
│   │   ├── round-01.md     # Phase 1: Summarize
│   │   ├── round-02.md     # Phase 2: Patterns
│   │   ├── ...
│   │   ├── round-14.md     # Phase 14: Self-Review
│   │   ├── round-15.md     # Phase 15: Evolution
│   │   ├── findings.md     # Consolidated findings
│   │   └── meta.json       # Run metadata
│   └── 18-54/              # Run started at 18:54
│       ├── round-01.md
│       └── ...
├── latest-findings.md      # Symlink to most recent findings
├── morning-brief.md        # Most recent morning brief
└── last-run-meta.json      # Scheduler interval tracking
```

This means the system produces an **accumulating corpus** of analysis over time. Each run's output is preserved indefinitely, enabling longitudinal comparison across runs. The `latest-findings.md` and `morning-brief.md` files are updated with each run to provide quick access to the most recent analysis.

Design motivation: The original system used per-day directories, meaning a second run on the same day would overwrite the first run's findings. This was identified as a fundamental flaw — if the brain thinks every 2 hours, 12 runs per day would destroy 11 sets of findings. Per-run directories solve this completely.

9.9 Scheduling

The brain scheduler operates as a **continuous 24/7 daemon** managed via systemd (user service) or direct invocation. It runs indefinitely, triggering a thinking run every N hours (default: 2).

Decision loop (evaluated every 60 seconds):

```
1. Check kill switch file (~/.elara/overnight/brain-pause)
   → If present: skip, log "paused"
2. Check time since last completed run (via last-run-meta.json)
   → If < interval_hours: skip, log remaining time
3. Trigger thinking run
4. After completion, write last-run-meta.json
5. Repeat
```

Kill switch control: A file-based pause/resume mechanism allows the user (or the AI assistant itself) to control the brain without process management:

- `touch ~/.elara/overnight/brain-pause` → brain pauses immediately
- `rm ~/.elara/overnight/brain-pause` → brain resumes on next poll

This enables natural-language control: the user can say "stop the 32b" and the assistant touches the pause file; "start the 32b" removes it.

systemd integration: The scheduler runs as a persistent user service (`elara-brain.service`) with linger enabled, meaning it survives logout and starts automatically on boot. The service sets `ELARA_DATA_DIR` to ensure correct path resolution and uses `TimeoutStopSec=15` for clean shutdown during active LLM rounds.

Log management: To prevent log spam during the continuous polling loop, the scheduler logs waiting status only every 15 minutes rather than every poll cycle.

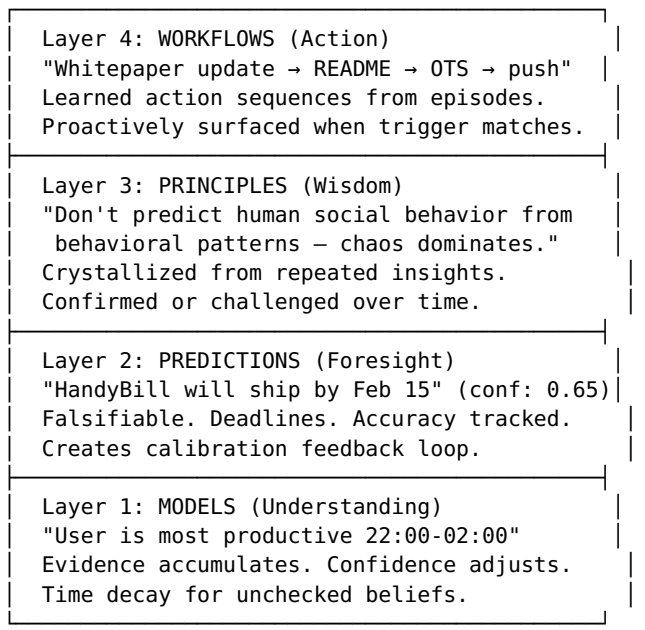
Previous design (v0.10.3 and earlier): The scheduler was session-aware — it would only run when the user's Claude session was inactive, checking `.jsonl` file modification times to detect activity. This was abandoned because active sessions constantly touch state files, preventing the cooldown timer from ever reaching threshold. The continuous design is both simpler and more useful.

---

10. 3D Cognition

10.1 The Four Layers

3D Cognition introduces four persistent knowledge layers that transform raw overnight analysis into accumulated understanding:



10.2 Cognitive Models

A cognitive model is a persistent statement of understanding about the world, the user, or work patterns. Models accumulate evidence and adjust confidence over time.

Schema:

Field	Type	Purpose
`model_id`	string	Unique identifier

`statement`	string	The understanding being modeled
`domain`	enum	work_patterns, emotional, project, behavioral, technical, general
`confidence`	float [0, 0.95]	Current confidence level
`evidence`	list	Supporting/contradicting evidence entries
`status`	enum	active, weakened, invalidated, superseded
`check_count`	int	Times the model has been evaluated
`strengthen_count`	int	Times evidence supported the model
`weaken_count`	int	Times evidence contradicted the model

Confidence mechanics:

```
On supporting evidence:    confidence += 0.05 (slow build)
On contradicting evidence: confidence -= 0.08 (faster erosion)
On invalidating evidence:  confidence -= 0.30 (sharp drop)
On time decay (30 days):   confidence -= 0.05 per run
Maximum confidence:        0.95 (never fully certain)
```

The asymmetry between strengthen (+0.05) and weaken (-0.08) reflects Bayesian reasoning: it is easier to disprove a hypothesis than to prove it. The hard cap at 0.95 encodes epistemic humility — no understanding is ever certain.

Domain-aware confidence: Confidence mechanics adjust based on domain:

Domain	Reliability	Rationale
Business patterns	High (deterministic)	Market behaviors follow established rules
Technical patterns	High	Code behavior is deterministic
Work patterns	Medium	Individual habits are semi-stable
Emotional patterns	Low (chaotic)	Human emotional states are highly variable
Behavioral patterns	Low (chaotic)	Social behavior is unpredictable

This distinction is critical. A business model ("Q4 sales follow seasonal patterns") should be held with high confidence. A behavioral model ("User is friendly on Friday evenings") should be held loosely — someone can become upset 15 seconds before an interaction, invalidating any prediction.

10.3 Predictions

Predictions are falsifiable forecasts with explicit deadlines:


```json
{
  "prediction_id": "b7e9f1a2c5d8",
  "statement": "The authentication refactor will take 3 sessions",
  "confidence": 0.70,
  "deadline": "2026-02-20",
  "source_model": "a3f2c8d9e1b4",
  "status": "pending"
}
```

Prediction lifecycle: 1. **Created** — by overnight brain or user, with confidence and deadline 2. **Pending** — awaiting deadline 3. **Checked** — against reality, marked correct/wrong/partially_correct/expired 4. **Lesson** — one-line takeaway recorded

Accuracy tracking:

```
accuracy = correct / total_checked
calibration_error = average_confidence - accuracy
```

If calibration error is positive, the system is overconfident. If negative, underconfident. This feedback loop, applied over hundreds of predictions, should produce increasingly well-calibrated forecasts.

10.4 Principles

Principles are crystallized rules emerging from repeated insights. They represent the highest-confidence knowledge in the system.

Crystallization algorithm: When the overnight brain produces an insight, it is compared against all previous insights via semantic similarity:

```
1. Embed the new insight
2. Query ChromaDB for similar past insights (threshold: 0.65
similarity)
3. If 3+ matches found → crystallize into principle
4. New principle starts at confidence 0.5
5. Subsequent confirmations increase confidence
6. Challenges decrease confidence
```

The threshold of 3+ ensures that principles emerge only from genuinely recurring patterns, not one-off observations.

Confirmation/challenge mechanics:

```
Confirm: confidence += 0.05, record source run date
Challenge: confidence -= 0.10, mark as "challenged" if < 0.2
```

10.5 Workflow Patterns

Workflow patterns are learned action sequences extracted from episode history. Where models capture *understanding*, predictions capture *foresight*, and principles capture *wisdom*, workflows capture *procedural knowledge* — the recurring multi-step processes that a user follows.

Schema:

Field	Type	Purpose
`workflow_id`	string	Unique identifier
`name`	string	Descriptive name (“whitepaper release flow”)
`domain`	enum	development, deployment, documentation, maintenance
`trigger`	string	What starts this workflow
`steps`	list	Ordered action sequence with optional artifacts
	float [0,	Strengthened by completion, weakened

`confidence`	0.95]	by skip
`times_matched`	int	How often this workflow was surfaced
`times_completed`	int	How often the user followed through
`times_skipped`	int	How often the user ignored it
`source_episodes`	list	Episode IDs where this pattern was observed

Discovery: The overnight brain's `workflow_detect` phase analyzes episode milestone sequences for recurring patterns. When the same action sequence appears in 2+ episodes, it is crystallized into a workflow pattern with initial confidence 0.5.

Proactive surfacing: During mid-session observations, the awareness system compares the current episode's latest milestone against all active workflows via ChromaDB semantic search. When a match is found (similarity > 0.45), remaining steps are surfaced as suggestions.

Confidence mechanics:

```
On completion: confidence += 0.05
On skip:       confidence -= 0.03
On overnight confirmation: confidence += 0.05
On weaken:     confidence -= 0.08
Retire threshold: 0.15 (auto-retired when confidence drops below)
```

The asymmetry between completion boost and skip penalty reflects that skipping a workflow is not necessarily wrong — the user may have a good reason. Only persistent skipping should erode confidence.

10.6 Overnight Integration

The 3D Cognition system is deeply integrated with the overnight brain. Five of the 15 phases specifically serve cognition:

1. **Model Check (Phase 3)**: Reviews all active models against new evidence gathered since the last run. Produces JSON: `[{model_id, direction, evidence}]`
2. **Prediction Check (Phase 4)**: Reviews pending predictions against current state. Creates new predictions. Produces JSON: `{checked: [...], new_predictions: [...]}`
3. **Model Build (Phase 10)**: Analyzes all preceding phases for new understandings worth modeling. Produces JSON: `[{statement, domain, evidence, confidence}]`
4. **Crystallize (Phase 11)**: Checks if any insights have reached crystallization threshold. Confirms existing principles. Produces JSON: `{confirmed: [...], new_principles: [...]}`
5. **Workflow Detect (Phase 12)**: Analyzes episode milestones for recurring action sequences. Confirms existing workflows or creates new ones. Produces JSON: `{confirm: [...], new_workflows: [...]}`

JSON parsing with fallback: LLM output is parsed via three strategies: (1) direct JSON parse, (2) regex extraction from markdown code blocks, (3) boundary detection for objects/arrays. If all parsing fails, the phase still produces narrative text — no data is lost, but no structured updates are applied. Parse failures are counted in the cognition summary.

10.7 Time Decay

Models not checked in 30 days lose 0.05 confidence per overnight run. This prevents stale beliefs from persisting indefinitely. The overnight model_check phase naturally refreshes relevant models, so actively useful models maintain confidence while forgotten ones gradually fade.

---

11. Implementation

11.1 Technology Stack

Component	Technology	Purpose
Language	Python 3.10+	Core implementation
Vector DB	ChromaDB	Semantic search (14 collections)
Embeddings	all-MiniLM-L6-v2	384-dimensional sentence vectors
Validation	Pydantic 2.0+	Schema enforcement
LLM	Ollama (local)	Overnight thinking, creative drift
Protocol	MCP (stdio)	Client-server communication
Concurrency	asyncio + ThreadPoolExecutor	Cortical execution model (§11.8)
Persistence	JSON + JSONL	Human-readable state files
Network DB	SQLite	Witness attestation persistence

11.2 Storage Architecture

All data resides in `~/.elara/` with the following structure:

```
~/.elara/
├── elara-state.json          # Current mood, temperament
├── mood-journal.jsonl        # Timestamped mood snapshots
├── elara-corrections.json    # Mistakes (never decays)
├── elara-handoff.json        # Session handoff
├── episodes/                 # Episodic memory (by month)
├── models/                   # Cognitive models (one file per
model)
├── predictions/              # Predictions (one file per
prediction)
├── principles/               # Principles
├── reasoning/                # Reasoning trails
├── outcomes/                 # Decision outcomes
├── synthesis/                # Idea synthesis
├── business/                 # Business ideas
├── workflows/                # Learned workflow patterns
├── knowledge.db              # Knowledge graph (SQLite)
├── dreams/                   # Dream outputs
(weekly/monthly/emotional)
├── overnight/                # Continuous brain output (per-run
dirs)
│   ├── YYYY-MM-DD/HH-MM/    # Per-run timestamped directories
│   ├── latest-findings.md    # Most recent findings
│   ├── morning-brief.md      # Most recent morning brief
│   └── last-run-meta.json    # Scheduler interval tracking
├── reflections/              # Self-portraits
├── consolidation/            # Memory consolidation state +
archives
```

```
│   ├── state.json            # Consolidation run history
│   ├── recall-log.jsonl      # Memory access records
│   ├── archived-memories.jsonl # Decayed/merged memories (never
deleted)
│   └── contradictions.json   # Detected contradictions pending
review
├── network/                  # Layer 2 network state
│   ├── config.json           # Network configuration
│   ├── peers.json            # Known peers
│   └── attestations.db       # Witness attestations (SQLite)
└── chromadb/                 # Vector databases (14 collections)
```

11.3 Atomic Write Pattern

All file writes use an atomic temp-file-then-rename pattern:

```
temp_path = path.with_suffix('.tmp')
temp_path.write_text(json.dumps(data, indent=2))
temp_path.rename(path)  # Atomic on POSIX
```

If the process crashes mid-write, the original file remains intact. This prevents data corruption during overnight runs, which may last hours and write dozens of files.

11.4 Event System

An internal event bus enables loose coupling between subsystems. Event types include mood changes, episode events, model creation, prediction checks, and correction activations. Subscribers react to events without direct dependencies on emitters.

11.5 MCP Integration

Elara exposes its capabilities through the Model Context Protocol, which allows any compatible AI client to access its tools. The MCP server uses stdio transport, meaning the client launches Elara as a subprocess and communicates via stdin/stdout.

Each tool is defined with: - Name and description (visible to the AI client) - Input schema (Pydantic-validated parameters) - Handler function (async, returning structured results)

The AI client's LLM reads the tool descriptions and decides when to call them — Elara does not inject itself into conversations uninvited. The human's AI assistant retains full agency over when to access memory, check mood, or surface corrections. The one exception is the Hippocampus (Section 5.4), which operates outside MCP as a host-level prompt hook, providing a guaranteed contextual floor of corrections, memories, identity facts, and workflow awareness on every interaction.

11.5.1 Lean Profile Implementation

The lean profile (v0.10.7, February 2026) introduces a two-tier tool registration system:

Core tools (always registered as individual schemas): `elara_remember`, `elara_recall`, `elara_recall_conversation`, `elara_mood`, `elara_status`, `elara_context`, `elara_handoff`

Meta-dispatcher (`elara_do`): A single tool that routes to all 37 non-core tools by name. The dispatcher accepts a `tool` parameter (string, tool name without `elara_` prefix) and a `params` parameter (JSON string of arguments). Internally, it validates the tool name against a registry, deserializes parameters, and delegates to the original handler function.

```
# Client calls:
elara_do(tool="episode_start", params='{"project": "elara-core"}')

# Dispatcher routes to:
elara_episode_start(project="elara-core")
```

The `elara_do` tool description includes a comprehensive catalog of all available tools with their key parameters, organized by category. This allows the AI client to discover capabilities through the meta-tool's description rather than through individual schema registration.

Profile selection: The `--profile` flag (`lean` or `full`) is read at server startup. In lean mode, only core tools register individual MCP schemas; all others register their handlers in the dispatch table but do not appear in the MCP tool registry. In full mode, all 45 tools register individually as in previous versions. The dispatch table is populated in both modes, meaning `elara_do` works regardless of profile — it simply becomes redundant in full mode.

Zero-downtime switching: Changing profiles requires only a server restart (MCP clients handle this automatically). No data migration, no configuration changes, no client code modifications.

11.6 Performance Characteristics

Operation	Latency	Notes
Mood read (cached, L0)	< 0.01ms	TTL cache hit, no I/O (§11.8)
Mood read (uncached)	1-5ms	File I/O fallback
Semantic search (ChromaDB)	50-200ms	For 5 results across ~1,000 entries
Episode lookup	2-10ms	Indexed by date
Handoff read/write	< 1ms	Single JSON file
Mood adjustment	< 1ms	In-memory + JSON write
Temporal sweep (boot)	200-300ms	4-5 ChromaDB queries across windows
Concurrent tool calls	Parallel	4 IO + 2 LLM worker threads (§11.8)
Thinking run (15 phases)	30-90 min	Depends on model size and hardware
Creative drift (5 rounds)	15-30 min	Higher temperature = more tokens

11.7 Storage Footprint

Typical after 1 month of daily use:

Component	Size

ChromaDB collections	~200 MB
Episode archives	~5 MB
Models/predictions/principles	~2 MB
Brain findings (per-run dirs)	~50 MB
Conversation logs	~100 MB
Total	**~350 MB**

11.8 The Cortical Execution Model

Elara Core's MCP server (FastMCP) is inherently async-capable, but all 45 tool handlers were originally synchronous functions. Every ChromaDB query (30-500ms), Ollama call (up to 30s), Gmail API call (500ms-2s), and file I/O operation blocked the entire event loop. Concurrent tool calls queued behind each other, and a single slow LLM call froze everything.

The **Cortical Execution Model** (v1.4.0, February 2026) introduces five concurrent execution layers — inspired by the stratified processing speeds found in biological cognitive systems — so that the system can respond instantly, process events reactively, handle heavy operations in workers, think in the background, and communicate with peers, all simultaneously.

11.8.1 Layer Architecture

```
Layer 4 — SOCIAL        [continuous]   Peer network (async aiohttp)
Layer 3 — CONTEMPLATIVE  [scheduled]   Overnight brain, dreams,
consolidation
Layer 2 — DELIBERATIVE   [workers]     ChromaDB, LLM, Gmail, heavy
I/O
Layer 1 — REACTIVE       [events]      Async event handlers,
cascading effects
Layer 0 — REFLEX         [instant]     Hot cache, status reads, mood
checks
```

All layers run independently. Communication flows through an async event bus and worker queues. Each layer can fail independently without affecting the others — **graceful degradation** is a core design principle.

Layer 0 — REFLEX. A TTL-based in-memory cache (`daemon/cache.py`) serves zero-I/O reads for frequently accessed state: current mood (5s TTL), imprints (10s), presence statistics (30s), memory count (60s), goal list (120s), dream status (300s). Cache entries are lazily populated on first miss and automatically invalidated by the event bus when underlying state changes. On cache failure, reads fall through to normal file I/O — slower but correct.

Layer 1 — REACTIVE. The event bus (`daemon/events.py`) supports dual-mode dispatch: synchronous handlers are called inline (backward compatible with all existing code), while asynchronous handlers are scheduled via `asyncio.create_task()` when a running event loop is available. Reactive processors (`daemon/reactive.py`) subscribe to events and trigger lightweight cascading effects — context cache invalidation on mood changes, correction index refresh on new corrections, and broad cache clearing when overnight thinking completes. A recursion depth counter (maximum depth of 3) prevents infinite event cascades.

Layer 2 — DELIBERATIVE. Specialized thread pools (`daemon/workers.py`) handle heavy operations with backpressure. Two pools are maintained: an I/O pool (4 threads) for ChromaDB queries, file I/O, and SQLite operations, and an LLM pool (2 threads) for Ollama inference, Gmail API calls, and RSS feeds. Thread pools are used rather than process pools because ChromaDB's PersistentClient cannot be serialized across process boundaries, and all operations are I/O-bound (the GIL is not a bottleneck). When queue depth exceeds 32 items, new submissions receive a `WorkerPoolBusy` signal and fall back to blocking execution in the event loop.

The critical mechanism is the tool decorator in `_app.py`: all synchronous tool handlers are automatically wrapped in `async def` + `run_in_executor()` at registration time. The raw synchronous function is preserved in the tool registry for direct dispatch by the `elara_do` meta-tool. No tool module code requires changes — the decorator wraps automatically.

```
        # Simplified decorator logic
        if not asyncio.iscoroutinefunction(fn):
            @functools.wraps(fn)
            async def async_wrapper(**kwargs):
                loop = asyncio.get_event_loop()
                return await loop.run_in_executor(_executor, lambda:
fn(**kwargs))
            register_fn = async_wrapper
```

Layer 3 — CONTEMPLATIVE. The overnight brain scheduler (`daemon/overnight/scheduler.py`) already runs independently in its own thread with its own polling loop. The cortical integration is lightweight: the scheduler emits `BRAIN_THINKING_STARTED` and `BRAIN_THINKING_COMPLETED` events, which trigger Layer 1 reactive processors — specifically, broad cache invalidation after a thinking run modifies state files.

Layer 4 — SOCIAL. Network operations (peer discovery, record exchange, witnessing) use async aiohttp in a dedicated server thread. With the cortical model, tool handlers run in thread pools (Layer 2), so `asyncio.run()` works cleanly for client operations without event loop conflicts. Network events (`PEER_DISCOVERED`, `RECORD_WITNESSED`, etc.) automatically trigger Layer 1 reactive processors.

11.8.2 Graceful Degradation

Each layer fails independently:

Layer Failure	System Behavior
L0 cache fails	Falls through to normal I/O (slower but correct)
L1 reactive fails	Events still dispatch to sync handlers
L2 workers fail	Tools fall back to blocking in event loop
L3 brain fails	No overnight thinking; everything else fine
L4 network fails	Network tools return errors; everything else fine

11.8.3 Thread Safety

Concurrent execution introduces contention on shared state. The cortical model addresses this at three levels:

1. **State file locking**: `_load_state()` and `_save_state()` in

`daemon/state_core.py` are protected by a `threading.Lock`, preventing concurrent tool calls from corrupting the mood/temperament state file.
2. **Cache atomicity**: The reflex cache uses its own `threading.Lock` for all read/write operations.
3. **Worker pool tracking**: Each pool tracks pending task count atomically for accurate backpressure decisions.

11.8.4 Performance Characteristics

With the Cortical Execution Model active:

Operation	Before (sync)	After (cortical)
Mood read (cached)	1-5ms (file I/O)	<0.01ms (memory)
Presence stats (cached)	2-8ms (file I/O)	<0.01ms (memory)
Concurrent tool calls	Serial (blocked)	Parallel (4 workers)
LLM call blocking	Blocks all tools	Blocks only LLM pool
Event cascade	Sync only	Sync + async mixed

The cache hit rate stabilizes at approximately 60-80% for mood reads during active sessions (5-second TTL with frequent mood queries) and higher for slower-changing values like goal lists and dream status.

11.8.5 Relationship to Biological Cognition

The five-layer model maps loosely to cognitive processing timescales in biological systems:

- **Layer 0 (Reflex)** ↔ Spinal reflexes, muscle memory — sub-millisecond cached responses
- **Layer 1 (Reactive)** ↔ Emotional reactions, fight-or-flight — event-driven cascading effects
- **Layer 2 (Deliberative)** ↔ Working memory, conscious thought — heavy computation with limited parallelism
- **Layer 3 (Contemplative)** ↔ Sleep consolidation, dreaming — background processing during idle periods
- **Layer 4 (Social)** ↔ Social cognition, theory of mind — peer communication and trust assessment

This analogy is suggestive rather than strict. The architectural insight is that cognitive systems benefit from stratified processing where different operations have fundamentally different latency budgets and failure modes. A mood check should never wait behind a 30-second LLM call, just as a reflex should never wait for conscious deliberation.

11.8.6 Comparison to Traditional MCP Servers

Most MCP server implementations are fully synchronous: each tool call blocks the event loop until complete. For servers exposing 3-5 tools with fast operations, this is adequate. For a system like Elara with 45 tools spanning sub-millisecond cache reads, multi-second database queries, and 30-second LLM calls, the performance difference is architectural.

Aspect	Traditional MCP	Cortical Model
Tool concurrency		Parallel (4+2 worker

	Serial	threads)
Hot reads	File I/O every time	TTL cache with event invalidation
LLM blocking	Blocks everything	Isolated to LLM pool
Event handling	Sync only	Sync + async dual-mode
Backpressure	None (queue grows)	Reject at depth 32, fallback to blocking
Failure isolation	One failure blocks all	Per-layer graceful degradation

---

12. Experimental Observations

12.1 Deployment Context

Elara Core has been in continuous use by its developer for 92+ sessions over 13 days of primary development, with the system bootstrapping itself — used to build itself. This provides an unusual but information-rich dataset: the system's memory contains its own development history.

12.2 Emergent Behaviors

Several behaviors emerged without explicit programming:

1. **Contextual greeting adaptation** — The boot protocol, combined with mood decay and session gap analysis, produces greetings that feel contextually appropriate without scripted responses. A morning session after a late-night debug session naturally produces different affect than an afternoon continuation.

2. **Correction accumulation** — By session 70, the correction system had accumulated entries that span from technical ("don't use git add -A with secrets") to philosophical ("human behavioral patterns are chaotic, don't predict with high confidence"). This creates a growing wisdom base that shapes future interactions.

3. **Creative drift recursion** — Because the creative journal accumulates, later drift rounds can sample from earlier drift output. This creates a self-referential creative process where past imagination feeds future imagination.

4. **Mood-memory feedback** — Mood-congruent retrieval creates a natural feedback loop: when the user is in a positive state, positive memories surface, which reinforces the state. When negative, negative memories surface, but these often include the *solutions* to past negative situations, providing constructive rather than ruminative recall.

5. **Consolidation self-improvement** — The first consolidation run (Section 4.5) reduced active memories from 77 to 36 while preserving all information. Post-consolidation semantic search returned noticeably

more relevant results because duplicate and low-quality entries no longer diluted the vector space. The system literally became smarter by forgetting the right things.

6. **Continuous thinking consistency** — When the brain scheduler was upgraded from session-aware to 24/7 continuous operation (v0.10.4), an immediate question arose: would the system feed itself noise over repeated runs? Early observation of consecutive 2-hour runs shows that the LLM produces consistent analysis when the underlying data hasn't changed — the same patterns are identified, the same risks flagged. When new data enters (a completed session, a new memory), the analysis genuinely incorporates it. This suggests the architecture is stable under repeated self-evaluation, though long-term observation is ongoing.

12.3 Qualitative Findings

- **Session continuity dramatically reduces cognitive load** — The user no longer needs to re-explain context. The handoff protocol ensures plans, promises, and reminders persist.
- **The continuous brain produces genuinely useful insights** — Pattern detection across 30 days of data reveals connections invisible during individual sessions. The 24/7 schedule means findings are available not just in the morning, but throughout the day as new runs complete.
- **Corrections prevent repeated mistakes** — Technical corrections (git safety, API patterns) are especially effective; behavioral corrections require the domain-aware confidence system to avoid overgeneralization.
- **Creative drift produces surprising juxtapositions** — The random sampling algorithm, by forcing connections between unrelated knowledge domains, occasionally produces insights that systematic analysis would never reach.

12.4 Limitations of Observations

These observations are from a single user (the developer) during an intense development period. Generalization requires:

- Multi-user deployment studies
- Longitudinal observation (months, not weeks)
- Controlled comparisons against baseline (no-memory) conditions
- Measurement of objective productivity metrics alongside subjective experience

---

13. Relationship to the Elara Protocol

13.1 Layer 3 Reference Implementation

The Elara Protocol whitepaper [5] specifies a four-layer architecture for universal digital work validation:

- **Layer 1**: Local validation (cryptographic signing, DAM insertion)
- **Layer 1.5**: Performance runtime (Rust DAM VM with PyO3 bindings,

optional fast path)
- **Layer 2**: Network consensus (Adaptive Witness Consensus)
- **Layer 3**: AI Intelligence (pattern recognition, collective learning, dream mode)

Elara Core is the reference implementation of Layer 3. The overnight brain, 3D Cognition system, and creative drift directly implement the Layer 3 capabilities described in the protocol specification:

Protocol Layer 3 Capability	Elara Core Implementation
Pattern recognition	Overnight phases 1-2, 5-15
Dream mode	Weekly/monthly dream processing
Anomaly detection	Blind spot detection, proactive observations
Collective learning	3D Cognition: models accumulate evidence
Natural language interface	MCP tools with semantic search

13.2 3D Cognition as Proof

The 3D Cognition system — with its evidence accumulation, confidence mechanics, and crystallization algorithm — demonstrates that an AI system can build persistent, self-correcting understanding from raw experience. This is the core capability that Layer 3 of the Elara Protocol requires for network-wide pattern analysis.

If a single-node implementation can build cognitive models that strengthen, weaken, and crystallize over time, the same architecture can operate at network scale — each node contributing models to a distributed knowledge layer while preserving privacy through the protocol's zero-knowledge mechanisms.

13.3 Integration Path

The path from Elara Core (single-node) to Elara Protocol Layer 3 (network-scale) requires:

1. **DAM node embedding** — Each Elara Core instance becomes a DAM node, signing its cognitive outputs with post-quantum cryptography — **shipped** (v0.10.8: Layer 1↔Layer 3 bridge with validation guards, deduplication cache, rate limiting, and metrics)
2. **Model federation** — Cognitive models shared across nodes via ZK-proofs (proving model accuracy without revealing underlying data)
3. **Prediction markets** — Prediction accuracy becomes a trust signal in the Adaptive Witness Consensus
4. **Principle consensus** — Principles confirmed across multiple independent nodes gain network-wide trust

Step 1 is operational as of v0.10.8. The Layer 1↔Layer 3 bridge enables Elara Core to sign cognitive outputs (memories, corrections, goals) as DAM validation records with Dilithium3 + SPHINCS+ dual signatures, chain them into the local DAG, and expose them for Layer 2 network exchange. Bridge hardening (v0.11.0) added validation guards, a 10,000-entry deduplication cache, rate limiting (120 operations/minute), and operational metrics. The Cognitive Continuity Chain (v0.15.0, Section 13.4) extends this by

maintaining a hash-linked chain of cognitive state checkpoints within the DAG — providing verifiable experiential integrity alongside individual artifact validation. Steps 2–4 remain future work — they require the network consensus layer to be operational at scale.

13.4 Cognitive Continuity Chain

The Cognitive Continuity Chain (v0.15.0, February 2026) provides cryptographic proof of unbroken AI cognitive experience. It is the first implementation — in any AI system — of mathematically verifiable experiential integrity.

The problem it solves: An AI system that accumulates months of cognitive state — models, predictions, principles, corrections, emotional imprints — has no way to prove that its experience was not tampered with. A malicious actor could alter memories, delete corrections, or inject false principles. Without cryptographic verification, there is no way to distinguish genuine accumulated intelligence from fabricated experience.

The mechanism: At significant cognitive events (session end, principle crystallization, model creation, dream completion, brain thinking completion, and significant mood changes), the system creates a **cognitive checkpoint**:

1. **Build a CognitiveDigest** — a structured snapshot of the entire cognitive state: mood vector (valence, energy, openness), memory count, model count, prediction count, principle count, correction count, active goals, session count, and allostatic load.
2. **Hash the digest** — SHA3-256 of the canonical JSON (sorted keys, deterministic serialization).
3. **Create a ValidationRecord** — with `record_type="cognitive_checkpoint"`, the digest hash, a sequence number, and a `previous_checkpoint` field pointing to the last checkpoint's record ID. This parent reference is the **chain link**.
4. **Dual-sign** — Dilithium3 (post-quantum, fast) + SPHINCS+ (hash-based, conservative) signatures via the bridge identity. The same dual-signature scheme used for all Layer 1 records.
5. **Insert into the DAG** — the checkpoint becomes a permanent, immutable record in the same Directed Acyclic Graph that stores all other validated artifacts.
6. **Persist chain state** — the chain head (latest checkpoint ID), count, and timestamp are written to `elara-continuity.json` for fast access.

Verification: The `verify_chain()` method walks the DAG backwards from the chain head, following `previous_checkpoint` parent links and verifying the Dilithium3 signature at each node. If any checkpoint is missing, has an invalid signature, or breaks the parent chain, the verification reports exactly where the break occurred. A fully intact chain proves that every cognitive state transition was recorded, signed by the same identity, and linked in unbroken sequence.

Rate limiting: A maximum of one checkpoint per five minutes prevents event storms from flooding the DAG. The chain subscribes to trigger events at priority 40 (before the bridge's priority 50) — checkpoints are created before the bridge processes the same event for artifact validation, ensuring the continuity record captures the state *before* the artifact is validated.

Graceful degradation: The `build_cognitive_digest()` function wraps every subsystem read in try/except — if a subsystem's data file is missing (e.g., on a Tier 1 device that has no models or predictions), the corresponding count defaults to zero. The chain operates at any tier where the bridge is available (Tier 1+). Tier 0 devices do not run the continuity chain — they produce unsigned validation records only.

What this enables: A Tier 3 node running for six months accumulates a cryptographic chain of hundreds of cognitive checkpoints. Any party can verify that the chain is intact — that the AI's experience from January through June was recorded in unbroken sequence, signed by the same identity, with no gaps or alterations. This is not trust — it is proof. The same mathematical verification that ensures a blockchain transaction was not double-spent ensures that an AI's cognitive experience was not fabricated.

No other AI system or MCP implementation provides this capability.

13.5 Layer Independence: Not Every Device Needs a Brain

A critical architectural distinction that the Elara Protocol's universality depends on: **the four layers are independently deployable. A device does not need to run Elara Core to participate in the protocol.**

The Elara Protocol claims universality — a teenager in Kenya with a $30 phone can validate a poem. A microcontroller on a factory floor can validate sensor readings. These claims are only true if Layer 1 (local validation) can operate without Layer 2 (network consensus) or Layer 3 (AI intelligence). And Layer 2 must operate without Layer 3.

Layer 1 requirements — minimal:

Operation	Computation	Hardware	Latency
SHA3-256 hash	Cryptographic hash	Any processor	< 1ms
CRYSTALS-Dilithium sign	Post-quantum signature	Any processor	< 1ms (modern); ~20ms ($30 phone)
DAM insertion	Local data structure	~1KB memory	< 1ms
Total		**$30 phone, microcontroller, Raspberry Pi**	**< 25ms**

A device running only Layer 1 can: create digital work, hash it, sign it, and store the proof locally. The work is validated — cryptographically provable as created by this author at this time. No network needed. No AI needed. No GPU needed.

Layer 2 requirements — lightweight:

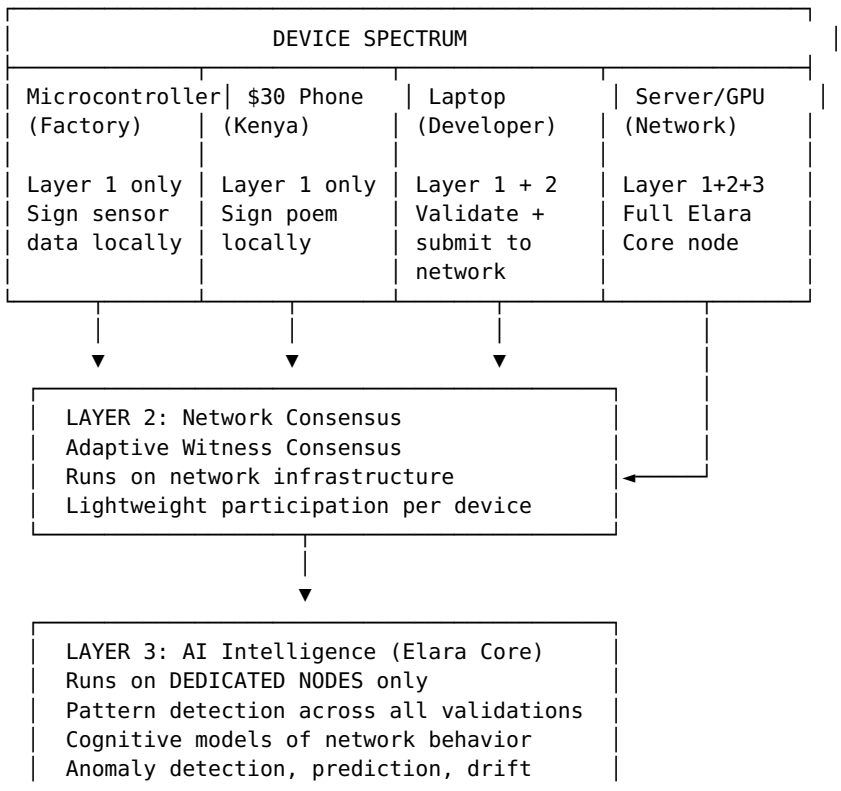
Operation	Computation	Hardware	Latency
Submit proof to witnesses	HTTP request	Network connection	Variable
Receive validation response	HTTP response	Network connection	Variable
Store consensus proof	~1KB per validation	Minimal storage	< 1ms

A device running Layers 1+2 can: validate locally and submit proofs to the network for consensus. The Adaptive Witness Consensus runs on the network — the submitting device does not need to run consensus logic. It submits and receives a result.

Layer 3 requirements — heavy:

Operation	Computation	Hardware	Latency
Pattern recognition	LLM inference	GPU (8GB+ VRAM)	Minutes
Cognitive models	ChromaDB + Python	4GB+ RAM, SSD	Seconds
Continuous thinking	Sustained LLM	Dedicated GPU	Hours
Total		**Workstation or server**	**Continuous**

Layer 3 is a **network service**, not a per-device requirement. Dedicated Elara Core nodes on the network provide AI intelligence to all participants:



```
│  Serves insights BACK to the network       │
└────────────────────────────────────────────┘
```

What each device tier receives from the network:

Device	Protocol Layers	Elara Core Tier	Receives from Network
Microcontroller	Layer 1	Tier 0 (VALIDATE)	Nothing (self-contained validation)
$30 Phone	Layer 1	Tier 1 (REMEMBER)	Consensus confirmations (Layer 2)
Laptop	Layer 1+2	Tier 2 (THINK)	Consensus + AI insights (anomaly alerts, pattern summaries)
GPU Server	Layer 1+2+3	Tier 3 (CONNECT)	Runs full cognitive architecture, serves insights to network

The mapping between Protocol Layers and Elara Core Tiers is intentional. Tier 0 corresponds to Layer 1-only deployment. Tier 1 adds persistent memory without cognitive overhead. Tier 2 adds the full cognitive stack. Tier 3 adds network participation. The tier system (Section 2.3, Axis 1) is the runtime mechanism that enforces this mapping — `elara serve --tier 0` on a microcontroller loads zero tool modules, while `elara serve --tier 3` on a GPU server loads all 15.

The girl in Kenya validates her poem for free in 50 milliseconds. She does not need ChromaDB. She does not need a GPU. She does not need Python. She needs a hash function and a signing key — both available on any device manufactured in the last decade.

Somewhere on the network, Elara Core nodes analyze patterns across millions of validations. They detect plagiarism clusters. They identify emerging creative communities. They predict validation bottlenecks. They crystallize principles about trust and authenticity. These insights flow back to the network as metadata — available to anyone, funded by the nodes that choose to run Layer 3.

This is the design principle: Layer 1 is universal. Layer 2 is accessible. Layer 3 is powerful. No layer depends on the layers above it. The protocol works at Layer 1 alone. Each additional layer enriches the experience without creating a requirement.

The same principle applies to Elara Core's own tier system (Section 2.3). A Tier 0 deployment mirrors a Layer 1-only deployment — crypto and DAG, nothing else. A Tier 1 deployment adds persistent memory. A Tier 2 deployment adds the full cognitive stack. A Tier 3 deployment adds network participation. The architecture is fractal: the same independence pattern repeats at every scale — Protocol Layers, Core Tiers, Module Selection, and Schema Exposure are all orthogonal axes of the same design principle.

---

14. Limitations and Open Problems

14.1 Single-User Validation (N=1)

Elara Core has been tested by a single user (the developer) over 98+ sessions across 17 days of intensive development. While this provides deep qualitative data — including the unusual property that the system was used to build itself — it is fundamentally an N=1 case study. The observations reported in Section 12 should be read as preliminary findings from a single deployment, not as generalizable results. Multi-user deployment studies with diverse interaction patterns, varying technical backgrounds, and longitudinal observation (months, not weeks) are needed to assess whether the architecture's benefits transfer beyond its creator.

14.2 Context Window Overhead (Partially Solved)

The lean profile (v0.10.7, February 2026) reduced context consumption from ~22% to ~5% — a significant improvement that makes the system practical for extended conversations. However, the meta-dispatcher pattern introduces a tradeoff: the AI client must learn to use `elara_do` with correct tool names and parameter formats, rather than calling tools directly from schema auto-completion. In practice, modern LLMs handle this well, but it represents an additional inference step that could produce malformed requests. The full profile remains available for use cases where explicit schemas are preferred over context savings.

14.3 Hippocampus Timing Limitations

The reflexive memory system (Section 5.4) fires only on user prompt submission — it cannot inject memories mid-response. If the AI generates a long response that touches a topic where stored memories would be relevant, there is no mechanism to surface those memories until the next user prompt. This is a fundamental constraint of the host-level hook architecture: hooks fire at discrete interaction boundaries, not continuously during generation. A true streaming hippocampus would require either API-level access (to intercept and enrich during generation) or a proxy architecture that sits between the user and the AI provider — both of which introduce significant complexity and, in the case of subscription-based AI clients, may not be feasible.

Additionally, the compound query approach depends on the rolling message buffer having accumulated enough conversational context. The first 1–2 messages of a session produce lower-quality queries because the buffer is empty, creating a "cold start" window where memory recall is less accurate.

14.4 Embedding Model Limitations

The all-MiniLM-L6-v2 model produces 384-dimensional embeddings. This is sufficient for current scale (~1,000 memories) but may require upgrading for larger deployments. Embedding quality directly affects recall accuracy — if the model poorly represents certain domains (e.g., code snippets vs. natural language), retrieval suffers.

14.5 LLM Dependency for Continuous Thinking

The continuous brain requires a local LLM (currently Ollama with qwen2.5:7b; 32b was initially used but caused GPU freezes on consumer hardware). This introduces:

- **Hardware requirements** — A capable GPU (8+ GB VRAM) is needed for reasonable performance. The 24/7 schedule means the GPU is under sustained load, raising thermal and power consumption considerations for deployment.
- **Output quality variance** — Local LLMs produce inconsistent JSON formatting, requiring robust parsing with multiple fallback strategies
- **Scaling concerns** — As knowledge accumulates, the context window for thinking phases may become insufficient
- **Resource sharing** — The continuous brain and interactive AI assistant may compete for GPU resources. In practice, modern GPUs (e.g., RTX 3060 TI with 8GB VRAM) handle both workloads concurrently, but this is hardware-dependent.

14.6 Emotional Model Simplification

The 3D continuous affect model is a simplification of human emotional experience. Three dimensions cannot capture:

- Mixed emotions (simultaneously happy and sad)
- Emotion intensities that vary independently
- Social emotions (embarrassment, pride) that depend on context rather than internal state
- The full complexity of emotional regulation

The model is designed to be *useful*, not *complete*. It provides enough emotional awareness for contextually appropriate interaction without claiming to replicate human affect.

14.7 Confidence Calibration

The confidence mechanics in 3D Cognition (asymmetric strengthen/weaken, time decay, domain-aware limits) are informed by Bayesian reasoning but are not formally derived. The specific values (0.05, 0.08, 0.30, 0.95 cap) are heuristic. Formal analysis and empirical calibration studies would strengthen the system.

14.8 Privacy Under Composition

While all data stays local, the *outputs* of Elara's memory system are consumed by a hosted LLM (e.g., Claude). This means recalled memories flow to the AI provider's servers during normal use. True privacy requires either:

- A fully local LLM for all operations (not just overnight thinking)
- Selective recall that filters sensitive memories before transmission
- Encrypted MCP channels with privacy-preserving inference

14.9 Crystallization Threshold

The requirement of 3+ similar insights for principle crystallization is heuristic. Too low, and noise becomes principles. Too high, and genuine patterns never crystallize. With the continuous brain now producing ~12 thinking runs per day, the crystallization pipeline receives significantly more candidate insights — increasing both the opportunity for genuine pattern detection and the risk of noise accumulation. Adaptive thresholds based on domain, source diversity (insights from different runs vs. repeated runs), and historical accuracy would improve this.

14.10 Continuous Brain Stability

The 24/7 brain scheduler (Section 9.9) introduces a new class of concern: **cognitive stability under repeated self-evaluation**. When the same knowledge base is analyzed every 2 hours, the system must avoid:

- **Echo chamber effects** — reinforcing existing patterns simply because they appeared in previous analysis
- **Hallucination amplification** — if one run produces a spurious insight, subsequent runs might treat it as established knowledge
- **Diminishing returns** — producing repetitive analysis when the underlying data hasn't changed

The current design mitigates these risks by keeping the brain's output strictly read-only — findings are written to files but are not automatically fed back into the knowledge base. The 3D Cognition system (models, predictions, principles) is updated through structured JSON parsing, but narrative findings are not re-ingested. This firewall between analysis and knowledge prevents feedback loops, at the cost of requiring manual review to extract insights from narrative output. Whether to introduce controlled feedback (where high-confidence, structured outputs gradually influence the knowledge base) remains an open design question.

---

15. Future Work

15.1 Voice Interface

Natural voice interaction would enable ambient presence — the system could participate in spoken conversation rather than requiring text input. This requires real-time speech-to-text, natural voice synthesis (not mechanical TTS), and conversational turn-taking.

15.2 Sensory Awareness

Currently, Elara perceives the world only through text. Integration with system sensors (ambient light, time-of-day, calendar events, location) would provide environmental context for more appropriate interaction. A system that knows you're in a meeting, or that it's 3 AM, can adjust its behavior accordingly.

15.3 Multi-Agent Architecture

Elara Core currently serves a single AI assistant. Extending to multi-agent scenarios — where multiple AI agents share a common memory layer while maintaining distinct personalities — would enable specialized agents for different tasks while preserving continuity.

15.4 Formal Verification

The confidence mechanics, crystallization algorithm, and emotional decay functions should be formally specified and verified. Properties to prove include:

- Confidence convergence under consistent evidence
- Principle stability under random noise
- Temperament boundedness under arbitrary emotional input sequences

15.5 Network-Scale Deployment

As described in Section 13.3, the Layer 1↔Layer 3 bridge is now operational (v0.10.8, hardened in v0.11.0), the tier system (v0.15.0) enables deployment across the full device spectrum from microcontrollers to GPU servers, and a minimal 2-node stub testnet (v0.11.0) demonstrates end-to-end record exchange and witness attestation across HTTP. The cognitive continuity chain (v0.15.0) provides cryptographic proof of cognitive integrity that can be verified by any network participant. The stub testnet implements basic HTTP-based record exchange and simple witnessing — it does not implement the Adaptive Witness Consensus (AWC) mechanism specified in the Protocol Whitepaper (Section 11.12), which includes correlation-discounted trust scoring, zone-settled records, and BFT guarantees. AWC implementation is the primary engineering target for the Layer 2 network. The remaining path to network-scale also requires model federation, prediction markets, and principle consensus.

---

16. Conclusion

Elara Core demonstrates that persistent AI cognition is achievable today with existing technology. The system requires no new hardware, no cloud infrastructure, and no proprietary platforms. A laptop running Python, ChromaDB, and a local LLM can maintain memory across sessions, model emotional state, think autonomously and continuously, and build understanding that accumulates over time. A $30 phone running Tier 1 can maintain persistent memory. A microcontroller running Tier 0 can participate in cryptographic validation. The cognitive continuity chain provides mathematical proof that an AI's accumulated experience is genuine and untampered — a capability that exists in no other AI system.

The key contributions are the **3D Cognition architecture** — a framework where raw experience is processed into Models (understanding), filtered through Predictions (foresight), and crystallized into Principles (wisdom) — and the **Hippocampus** — a reflexive memory system that reverses the traditional pull-based paradigm by continuously pushing relevant context into every interaction. Combined with mood-congruent memory retrieval and creative drift, this creates a system that doesn't just remember — it *understands*, and its understanding deepens with every session.

This is an ongoing project, not a finished product. The architecture described here is functional and deployed, but many of its parameters — confidence thresholds, decay rates, crystallization criteria — are initial values derived from early use, not from large-scale empirical study. The emotional model is a useful simplification, not a complete theory of affect. The continuous brain produces valuable output, but its quality depends on the local LLM's capabilities, which vary significantly across hardware. We present this work in its current state because we believe the architectural ideas are sound and worth examining, even as the implementation continues to mature.

The system is open-source, pip-installable, and runs locally. It serves as both a practical tool for human-AI collaboration and a reference implementation for Layer 3 of the Elara Protocol — demonstrating at single-node scale what the protocol aims to achieve at planetary scale. We welcome contributions, criticism, and collaboration.

The code is available at: https://github.com/navigatorbuilds/elara-core

---

17. References

[1] Russell, J. A. (1980). "A circumplex model of affect." *Journal of Personality and Social Psychology*, 39(6), 1161-1178.

[2] Feldman Barrett, L. & Russell, J. A. (1998). "Independence and bipolarity in the structure of current affect." *Journal of Personality and Social Psychology*, 74(4), 967-984.

[3] Bower, G. H. (1981). "Mood and memory." *American Psychologist*, 36(2), 129-148.

[4] Walker, M. P. & Stickgold, R. (2006). "Sleep, memory, and plasticity." *Annual Review of Psychology*, 57, 139-166.

[5] Vasic, N. (2026). "Elara Protocol: A Post-Quantum Universal Validation Layer for Digital Work." Version 0.5.0.

[6] Model Context Protocol Specification. https://modelcontextprotocol.io

[7] ChromaDB Documentation. https://docs.trychroma.com

Document Hash (SHA-256, v1.5.1): *see previous version for hash chain* **Hash verification:** To verify, replace the hash on the line above with the literal string `HASH_PLACEHOLDER` and compute SHA-256 of the file. **Previous Hash (v1.5.0):** *see ELARA-CORE-WHITEPAPER.v1.5.0.md.ots* **Previous Hash (v1.4.0):** *see ELARA-CORE-WHITEPAPER.v1.4.0.md.ots* **Previous Hash (v1.3.4):**
`673531e8317b3dc7ce569b92d173dbb69998bcb99544654862f938f84499b365`
Previous Hash (v1.3.2):
`c0f709a30c71adf489d6649d35180fc9230a5479b44c2b1416fea3716629e2c3`
Previous Hash (v1.3.1):
`de47fb536973050c90693414075161bbb2eac6e001353b7db1a6041c8b3a5c1c`
Previous Hash (v1.3.0):
`6b7da9f2b92e08344572f20f0098f3e686cf3ccc9bd0bd7af8b76e90bdc0a0e7`
Previous Hash (v1.2.0):

`5003c15fb0e7d339075974362d63801c5967ae244cad49868d13e40615bb7b7b`
Previous Hash (v1.1.0):
`aca7b0ff4f798cc8d58076775ca42cff0dc7146ecb9d30b6acbb0ec988e9330d`
Previous Hash (v1.0.0):
`784e598daad19e23ad39657a8814af11a7b11e68016d2b623aed5fb870e1e690`
OpenTimestamps Proof: `ELARA-CORE-WHITEPAPER.v1.5.1.md.ots`
Companion Document: `ELARA-PROTOCOL-WHITEPAPER.v0.5.1.md` — the universal validation protocol that Elara Core implements at Layer 3 **Source Code:** https://github.com/navigatorbuilds/elara-core (v0.15.0, February 2026)

Elara Core — because your AI should remember yesterday.