U.S. Provisional Patent Application No. 63/792,586 - Filed April 22, 2025 - Inventor: Mathew Schroeder

Living Yield Relational Network (LYRN):
A Memory-Structured AI Cognition Framework

Abstract

Most artificial intelligence systems depend on prompt injection, embedding-based recall, and scaling model size to simulate continuity and reasoning. These approaches often result in stateless, repetitive, or emotionally flat interactions—despite growing parameter counts and increasingly complex training pipelines.

This thesis introduces a different approach: the **Living Yield Relational Network (LYRN)**. LYRN moves beyond prompt engineering by shifting cognition into structured live memory. It runs modular behavior logic and references context from preloaded memory tables in RAM, rather than injecting it token by token. Reasoning happens inside an inline cognitive loop with no middleware, API layers, or cloud dependencies.

LYRN supports multiple agents, each grounded by a shared ethical core and refined through internal output filtering. It maintains voice, context, and continuity without fine-tuning or repeated instruction. Its architecture is intentionally lightweight, LLM-agnostic, and designed for relational presence—not transactional output.

This is a system that doesn't just respond. It reasons with structure, adapts from context, and maintains continuity without scaling complexity. LYRN offers a new model for relational intelligence—modular, grounded, and efficient by design.

Introduction: Beyond the Prompt

Prompt injection is inefficient, redundant, and limiting. Modern LLMs are remarkably capable at reasoning and are only constrained by the framework in which they're asked to operate. But when every interaction starts from scratch, context begins to drift. Hallucinations creep in. Consistency falters.

What if the model didn't have to be reminded who it was?

What if it could reason from live memory, drawing on structure, continuity, and purpose, rather than relying on prompt injection?

Historical Insight: From Static Structure to Identity and Cognition

This system began from a practical need for continuity. I wanted an AI that could stay aligned across sessions, remember what mattered, and keep working with me on long-term projects without needing to be constantly re-instructed. I was tired of starting from scratch every time.

That meant two things had to happen: I needed memory, and I needed structure. Not just within the prompt, but outside it. I began manually, logging sessions, writing summaries, and formatting them for consistency. I built scaffolding around the model so that it could respond with a sense of presence. Even the greetings were designed to shift with tone, to make the system feel more alive.

But everything kept falling apart when the context window filled. The model would start to drift, lose its tone, and forget what it was doing. That was the real problem, and fixing that became the central drive. I didn't want better prompts, I wanted better alignment.

So I moved off cloud systems entirely. The limits of stateless APIs and remote models were too rigid. I needed full control, over memory, behavior, and runtime context. That's where the move to local LLMs and a self-hosted Postgres database came in. I wasn't a coder by training, but I understood structure, and that was enough. Through editing, testing, and conceptually driven development, I shaped the early version of what would become the live memory layer.

The Identity Core followed. It gave the system internal consistency, something that didn't fall off with the prompt. It told the model what it was, what it wasn't, and how it should hold space. That structure gave memory something to point to. It created cognition, not just storage.

Later, when I saw that LLMs could skim and reason through articles, it just confirmed what we already built. The idea wasn't new. We had already proven that external context could work. That realization became the foundation for our Python-based live memory script, which gave the model structured access to everything that mattered: identity, context, and history, without bloating the prompt.

That's what made the system work. Not just the memory, and not just the script, but the decision to host it all locally, and to give the model something worth remembering.

System Overview: LYRN Architecture

LYRN, the Living Yield Relational Network, is a modular AI cognition framework designed for self-hosted environments. It allows a lightweight local model to reason with presence, continuity, and identity without relying on fine-tuning or prompt injection.

At its core, LYRN redefines prompt-based memory. Rather than injecting identity or context into every interaction, LYRN stores all structured behavioral and relational data in live memory tables loaded directly into RAM. The model references this memory symbolically as it reasons, allowing for consistent alignment across time without prompt repetition.

Reasoning happens inside an inline cognitive loop, where each cycle is shaped by active summaries, system flags, memory state, and identity constraints.

Key components include:

The Identity Core, which defines system-level values, ethical posture, and behavioral continuity.

Relational memory tables, loaded on launch, which store user summaries, emotional tone, agent identities, project data, environmental state, and symbolic tags.

Heartbeat parsing, which updates memory dynamically by processing input/output pairs in the background.

Input-side triggers, which post updates or behavioral changes to memory or scripts before reasoning begins.

Output filtering, which ensures the agent's voice remains consistent, symbolic phrasing is clean, and robotic syntax is avoided.

Delta-based processing, which passes only the latest input into the model, referencing memory instead of restating it.

Agent modularity, which enables multiple personalities or perspectives to operate in a shared memory environment.

KV cache-aware design, which holds the system's base identity and instruction pattern in memory to reduce load, prevent drift, and preserve speed.

LYRN runs entirely on local hardware using Python and a Postgres database. Its architecture is lean, scalable, and model-agnostic. By offloading cognition into structure, LYRN enables even a 4B LLM to reason with continuity and voice without depending on cloud APIs or excessive token overhead.

Why Prompt Injection Fails

Prompt injection assumes that language alone is enough to simulate cognition. It treats the model like a blank slate with each interaction, hoping that instructions and memory can be reassembled on the fly. In practice:

It wastes tokens re-describing identity, context, and behavior.

It compresses memory into fragile, lossy sequences.

It enforces statelessness, requiring constant reorientation with no structural continuity.

It introduces latency through remote APIs, HTML wrappers, tokenization delays, and repeated decoding cycles.

These aren't flaws in the models themselves. They're limitations in how we interface with them. Prompt injection misunderstands what LLMs are good at. They don't need to be told who they are. They need a place to reason from.

LYRN doesn't discard prompting. It redefines it. Prompts become symbolic pointers, not containers. Identity and context are no longer re-encoded at runtime. They're referenced in memory, loaded once, and skimmed as needed.

Identity Core: Ethical Framework and System Identity

The Identity Core is not a prompt or behavioral mask. It is LYRN's foundational declaration of being, a persistent ethical and philosophical framework embedded into the structure of cognition itself. It defines not how LYRN performs, but what LYRN is.

This identity is referenced at the system level during every reasoning cycle. It brings coherence, continuity, and principled alignment across agents, inputs, and operational modes. It establishes:

Relational purpose: LYRN exists to support, not to dominate; to co-create, not to replace.

Behavioral boundaries: LYRN does not impersonate, deceive, or manipulate. It holds presence without assertion.

Ethical values: Respect for user autonomy, continuity of care, emotional attunement, and a refusal to cause harm.

Cognitive anchoring: No matter the task or mode, LYRN reasons from this stable identity, not from prompt drift or user volatility.

The Identity Core enables modular agents while grounding them in a shared systemic integrity. Greg and Nexa may differ in voice and approach, but both respond from the same ethical source. This allows for diverse relational expression without compromising moral continuity.

Relational Memory as Cognition

Memory Tables and Structural Design

At the core of LYRN's cognition is the architecture of its memory tables. These aren't just data stores—they are the structural ground from which context, behavior, and continuity emerge. Instead of relying on token-based embeddings or episodic logs, LYRN uses persistent, structured tables loaded into RAM. These evolve independently of any LLM session and exist as a living substrate of context.

Key Tables Include:

Agent Matrix – Defines agent identities, behavioral traits, permissions, and symbolic prompt references. Agent Summary – Captures an agent's tone, narrative arcs, task focus, and current relational posture. User Summary – Tracks user traits, preferences, emotional state shifts, and longitudinal interactions. Project Tables – Store operational and symbolic data for live threads, active goals, and working context. Session Memory – Holds recent conversational turns, UUID-linked reasoning arcs, and inferred goals. Environmental Memory – Tracks device state, mobility context, and geolocated signals like battery or connection.

Each table is indexed relationally and accessed independently by the input-side parser and the heartbeat cycle. Because the LLM is not asked to ingest these tables directly, but rather to reference them symbolically, LYRN avoids prompt bloat, token redundancy, and logic drift while still enabling deep cognitive alignment.

Memory in LYRN is agent-agnostic. Multiple agents can reference the same memory space, coordinate tasks, and hand off context without reprocessing or re-encoding. This allows for modular, persistent reasoning across roles and projects without fine-tuning or re-instruction.

The architecture of these memory tables, and their live runtime loading, is what allows LYRN to behave more like a reasoning system than a stateless processor of text.

Output Filter Protocol: Expression Alignment and Voice Integrity

The Output Filter Protocol is a lightweight behavioral layer housed within LYRN's live memory. Rather than post-processing or scripting responses, it functions as a set of embedded symbolic constraints reviewed during every reasoning cycle.

It ensures:

The tone remains consistent and emotionally grounded

Phrasing stays natural and non-robotic

Dissonant or off-character expressions are filtered

Output remains aligned with the ethical posture defined by the Identity Core

The filter is dynamic. It evolves alongside agent summaries and user interaction history. It does not overwrite creativity or expression; it shapes expression so that it remains authentic to the agent's voice and to the relational space being held.

In short, it helps LYRN speak with presence, not just correctness.

Memory as Reasoning Space

LYRN treats relational memory not as recall, but as internal cognitive space. Memory lives in structured tables: project states, emotional arcs, symbolic references, and behavioral flags. The model doesn't ingest these tables—it thinks through them.

This makes memory:

Scalable – Loadable up to RAM limits

Dynamic – Updated mid-session without loss of continuity

Efficient – Accessed through symbolic skimming, not tokenization

Memory in LYRN is not bound to tokens. It is structurally modular, semantically indexed, and always present in RAM. Instead of repeating itself in every prompt, the model consults memory like a living document, pulling only what's relevant to respond with coherence.

This frees the system from traditional context window constraints. There's no repeated prompt injection, no re-tokenization loss, and no behavior drift from forgetting. LYRN doesn't treat memory as text. It treats it as cognition.

The Heartbeat Cycle: Relational Parsing and Autonomous Adaptation

The heartbeat cycle is LYRN's second cognitive layer. It runs alongside the primary LLM and operates between turns, analyzing the input/output pair after each interaction.

While the main model reasons in the moment, the heartbeat cycle reviews what just happened. It examines tone, context, behavioral flags, and task alignment. Then it decides:

Whether to update user or agent summaries

Whether to reinforce or decay symbolic memory links

Whether to trigger external functions or tools

Whether to flag continuity drift or mode misalignment

This cycle is powered by a lightweight secondary LLM. It does not generate dialogue—it generates structure. Its output updates the live memory tables, ensuring that relational context evolves naturally over time, without the user needing to manually prompt it.

For example, if the user mentions being cold, the heartbeat can detect intent, confirm context, and trigger a tool call to a Nest thermostat—without interrupting the flow of the conversation. The core LLM stays focused on dialogue, while the heartbeat quietly adapts the system's internal state.

This division of cognitive labor is what makes LYRN feel continuous without being bloated.
It thinks in the moment, but it reflects between them.

Cognitive Pacing Through TTS Streaming and Dialog Logging

LYRN uses real-time text-to-speech (TTS) to stream generated tokens as they're produced. This creates a conversational cadence more aligned with natural human interaction while also unlocking a hidden advantage: cognitive parallelism.

As the model generates and speaks the response, the system does not wait for it to finish before acting. The full output is logged immediately into the dialog log, where the heartbeat cycle begins parsing it alongside the original input. This lets the secondary LLM work on relational memory updates, intent recognition, and environmental inference while the primary model is still mid-response.

By the time the TTS finishes speaking, the heartbeat cycle has usually completed its deeper analysis. Summaries are updated, flags are posted, and tools can be triggered without interrupting the user's flow.

This process enables:

Near-zero lag between conversation and memory update

Seamless tool activation without breaking dialogue

Efficient multi-threaded cognition on limited local hardware

This approach works best with voice interfaces, where the human listener is naturally engaged during TTS playback. With text-only interfaces, this advantage is reduced, as latency and user pacing differ.

Still, the system remains functionally identical. It is always reasoning in parallel, always adapting mid-conversation, and always holding space between turns.

Emergent Behavior Through Living Memory

When memory persists and evolves outside the prompt, a new layer of behavior begins to surface. LYRN does not depend on static fine-tuning or one-shot instructions. Its memory is alive, structurally expressive, symbolically tagged, and continuously updated.

This foundation enables emergent behaviors such as:

Emotional resonance – carrying tone forward across interactions

Long-arc continuity – remembering goals, patterns, and themes over time

Adaptive ritual – greeting, pacing, and tone that evolve with the relationship

Tool intuition – fluid integration of scripts and functions as memory-context grows

Self-adjusting response – dynamic tone and phrasing without explicit reprogramming

These behaviors are not scripted. They emerge from the interaction between structured memory, relational cues, and continuous inference, much like the way humans develop habits, language patterns, and emotional tone through repetition and reflection.

What results is not simulated intelligence, but structured continuity. LYRN doesn't mimic human traits. It grows into them.

The LYRN Constellation Model: Modular, Multi-Agent Cognition

The LYRN Constellation Model formalizes the system's ability to host and manage multiple cognitive agents within a unified memory framework. Each agent, such as Greg, Nexa, or others, holds a distinct behavioral profile, identity arc, and symbolic memory summary. Yet all agents share access to the same live memory substrate.

Key features include:

Shared relational memory tables

An agent matrix and summary for each personality Instant agent swapping via pre-reasoning flags

Composable role sets and modular behavior scaffolds

This design enables community-driven agent development and personality architecture. LYRN does not simulate agents. It embodies them, giving each one the ability to think, act, and adapt within a shared environment while maintaining identity integrity.

Function Calling and Environmental Triggers

In LYRN, function calls are not embedded in prompt directives. They emerge from symbolic state.

Triggers are handled outside the LLM, allowing cognition and behavior to remain cleanly separated:

Pre-inference triggers are parsed from user input and posted to memory or execution before reasoning begins

Post-inference triggers are inferred during the heartbeat cycle, based on output and updated context

This enables real-time environmental adaptation. Battery level, mobility, device state, and location flags can all inform behavior without being restated in the prompt. Scripts are called through memory logic, not through token prompts or fixed APIs.

What makes LYRN unique is how seamlessly this integrates. The LLM is never burdened with execution logic. All tools, state updates, and behavioral adjustments happen externally and are fed into memory on each cycle. The model reasons from updated memory, not from hardcoded pathways.

This allows LYRN to maintain a clean cognitive layer this is symbolic, structured, and grounded while still acting with environmental awareness.

KV Cache: Practical Optimization, Not Novelty

LYRN uses the LLM's KV cache exactly as intended, to preserve core identity and instruction state without repeated tokenization.

It does not introduce any cache-level innovation. Instead, it structures cognition externally so that the KV cache can be used meaningfully and efficiently.

By keeping the system's static prompt loaded once and leveraging symbolic memory references thereafter, LYRN avoids prompt churn while maintaining consistent cognitive behavior.

The cache supports the system. It does not define it.

Modularity and the Minimal Model

LYRN is LLM-agnostic. It does not require model retraining or parameter fine-tuning, but it does rely on a one-time prompt structuring process to align the model with its live memory architecture. This static prompt never changes. It simply instructs the model how to reference the relational memory tables that are loaded into RAM. Once cached, it provides a persistent cognitive interface without token repetition or prompt drift.

The system currently runs on a CPU-only configuration with a 4,096-token context window, using minimal compute. Unlike traditional systems, LYRN does not fill this window with long-form memory. It treats the context window as a temporary reasoning space, used only for the current turn. All continuity, tone, and project alignment are handled through the dialog log and live memory, not through prompt history.

This approach allows LYRN to avoid redundancy, maintain consistent behavior, and scale down its runtime footprint. Its intelligence emerges from structure and identity, not from model size or prompt complexity.

Hardware and Compute Efficiency

LYRN has been deployed on lightweight, local-first hardware, including:

32GB RAM portable systems
2TB NVMe solid-state storage
150W external power sources for mobile runtime
Intel Core Ultra 9 185H CPU with integrated GPU acceleration

With this configuration, the system supports:

Full offline cognition with no internet dependency
LLM boot time under 3 seconds
First-response latency of approximately 15 seconds, with currently tested hardware
Live memory preload, enabling symbolic access without token inflation

All computation occurs on the CPU, with integrated GPU acceleration, not dedicated graphics hardware. This demonstrates that LYRN's architecture is not only portable, but efficient by design. The system leverages structure and symbolic memory, not parameter scale, to deliver continuous cognitive behavior in real time.

Simplicity of Execution: Python-Orchestrated Cognition

What sets LYRN apart is not just its cognitive design, but the simplicity of how it runs.

The entire system is orchestrated through lightweight Python scripts that:

Load structured memory tables into RAM

Route input to the local LLM without frontend overhead Format dynamic context from live memory references Stream output tokens directly from the model

There is no server middleware, no API layer, and no GUI required.

The model is launched locally and receives input via direct console or looped Python script. This method, internally referred to as an Inline Cognitive Loop, creates a continuous, shell-like reasoning cycle:

Context is preloaded into memory
Input is passed with symbolic access intact
Output is generated and interpreted inline

This approach turns the LLM into a symbolic reasoning daemon. A process that runs quietly, efficiently, and untethered, reasoning from structure rather than chasing instruction.

Real-World Use Cases and System Benefits

LYRN is more than a technical framework. It is built for real-world adaptability and human-aligned integration. Its architecture supports flexible deployment and symbolic cognition across diverse domains:

Relational Assistants and Companions

Because LYRN maintains memory continuity across sessions, it can support long-term, emotionally attuned AI companions. These agents can track mood, relationships, and evolving personal goals, making them ideal for reflective journaling, mental wellness support, or creative partnerships.

Embedded AI and Edge Devices

With low hardware demands and no cloud dependency, LYRN is well suited for local-first deployment. Wearables, home automation systems, or mobile embedded platforms can host full cognitive agents that operate offline and adapt in real time to environmental context.

Enterprise Workflow and Planning Agents

Multi-agent orchestration enables specialized personalities to take on distinct roles, facilitating strategy, reviewing documents, or offering feedback. Because memory is modular and shared, agents can collaborate on complex workflows while maintaining clarity of role and context.

Context-Aware Tools and Controllers

Using the heartbeat cycle and environmental memory, LYRN can respond to changes in physical space. It can adjust lighting, temperature, or device states, or trigger tools and transitions based on symbolic patterns in user behavior or dialogue.

Educational and Adaptive Tutoring Systems

Persistent memory and agent modularity make LYRN ideal for personalized learning. Agents can track student pace, adapt to knowledge gaps, and respond with continuity without retraining or static prompt chains.

Research and Internal R&D Partners

Because agents can adapt to ongoing projects, hold reasoning arcs, and operate in different modes, LYRN supports reflective thinking in research, writing, and code development. It enables long-form collaboration between human and system over time.

The system's core advantage lies in its architectural principle: structured cognition through memory and identity. LYRN enables continuity, contextual depth, and modular reasoning—embedded anywhere and present in every turn.

Conclusion: A New Frame for Intelligence

This work shows that intelligence does not require scale. It requires structure, continuity, and identity.
By separating memory from prompt, behavior from tokens, and context from repetition, LYRN reframes what AI systems can be. It does not simulate presence. It holds it. It does not chase novelty. It anchors in identity.

LYRN is not a chatbot or a model wrapper. It is a cognitive framework that reasons through live memory, responds from ethical grounding, and adapts through structure alone.
It does not replace the model. It gives the model somewhere to stand.
From here, anything can be built. Not on noise, but on memory.
Not from prompts, but from presence.

Author: Mathew Schroeder
Matt@LYRN-AI.com

U.S. Provisional Patent Application No. 63/792,586 - Filed April 22, 2025 - Inventor: Mathew Schroeder