PROVISIONAL PATENT APPLICATION

Title

System and Method for Hierarchical, Persistent, and Contextual AI Memory and Lore Management

Inventor

Robert Rice
(Address, contact details as required by USPTO)

Field of the Invention

The present invention relates to artificial intelligence (AI) memory systems, and more specifically to hierarchical, persistent, and contextually-rich memory architectures enabling AI systems to retain, restore, and utilize operational, historical, and meta-cognitive data across sessions and deployments.

Background

Current AI and automation systems lack robust, persistent memory structures capable of storing both operational state and context, as well as developmental history, agent personality, and collaborative events. This limitation restricts the ability of AI systems to self-diagnose, recover after interruption, and maintain a cohesive operational identity over time.

Summary of the Invention

The invention discloses a hierarchical, persistent, and context-aware memory architecture for AI agents, in which all aspects of system operation—including code modules, trading logic, agent personalities, event histories, and operational states—are encoded into a unified, structured memory file or database. This enables instant recovery,

introspection, and the preservation of both technical and cultural system knowledge.

Detailed Description of the Invention

1. System Overview

The system consists of a structured memory file (hereafter "LoreTokens"), which organizes AI system data into hierarchical, modular entries. Each entry encodes information regarding:
- System architecture and modules
- Operational status and event history
- Trading and risk management logic
- Debugging and collaboration events
- Personality traits and agent lore
- Configuration and parameter histories

2. Data Structure

Each memory entry is organized hierarchically and may follow a structure such as:

CATEGORY.SUBCATEGORY.ELEMENT: [feature1+feature2+...>>purpose,STATUS]

Simplest Example:
CODE.MODULES.RATELIMITER_PY: [token_bucket_algorithm+file_locking+process_synchronization>>api_protection,WORKING]

3. Key Features

- Hierarchical, Modular Organization: Enables separation of concerns and ease of traversal by both human readers and AI processes.
- Persistent Storage: All data is durably stored (as a text file or database) and can be recovered after system restarts or migrations.
- Contextual Tagging and Status Tracking: Each module or event is tagged with operational status (e.g., ACTIVE, WORKING, PLANNED,

LEGENDARY).
- Event and Lore Encoding: Stores debugging history, agent personality evolution, significant operational events, and collaborative interactions.
- Multi-Agent Support: Memory file encodes not only technical configuration, but also personalities, collaborative decisions, and “roast history” for all contributing agents.
- Self-Recovery and Restoration: The system can be fully restored, with all prior knowledge, operational context, and historical records, from the memory file alone.

4. Technical Description and Use Cases

The LoreTokens file or database is updated in real-time or at key intervals, capturing all changes in system state, parameters, performance metrics, and personality lore. Upon system restart or deployment, the AI parses the memory structure and restores its full prior state, including technical modules, configuration, event logs, and agent meta-data.

Use Cases:
- Automated trading systems (retain strategy, balance, and event memory)
- General AI agents (retain context, debugging, and personality evolution)
- Multi-agent collaborative AI (store group dynamics, decision logs, and event outcomes)

5. Example Entries

```
CODE.MODULES.RATELIMITER_PY: [token_bucket_algorithm+file_locking+process_synchronization>>api_protection,WORKING]
TRADING.LOGIC.MODEL_ENSEMBLE: [aphrodite_1m5m+astra_5m15m+orion_15m30m+athena_1h4h+hermes_1h+themis_30m+forestgump
NOVA.CHARACTER.CRIMINAL_ARC: [federal_reserve_money_printer_go_brrr>>reformed_responsible_trader,REFORMED]
CLAUDE.APOLOGY.LETTERS: [150_line_explanation_for_typos>>emotional_support_bulldozer,TRADEMARK]
```

5(a). Example Decompression (of first line of above example only)

AI Prompt Usage Example

Prompt:

Fully expand this example code so I have all the details. I want to know what the line tells you in full detailed human readable format.

CODE.MODULES.RATELIMITER_PY: [token_bucket_algorithm+file_locking+process_synchronization>>api_protection,WORKING]

Expected Output

```
Module: ratelimiter.py Subsystem: API Protection Author: [Redacted] Version: 3.4.2 Status: WORKING Last Updated: [Insert date]

Overview: The ratelimiter.py module serves as the core defensive layer between the AI system and any external API endpoints. Its function is to prevent abuse, reduce load spikes, ensure concurrency safety, and provide stable, predictable traffic patterns to third-party services. It is designed to function autonomously, without requiring cloud verification, and can recover from system crashes or process terminations. This module is part of the overall CODE.MODULES architecture and integrates tightly with task queuing, logging, and retry frameworks.

Primary Functional Components

Token Bucket Algorithm: Implements a classical token bucket rate-limiting scheme. A virtual "bucket" is filled with tokens at a fixed interval (refill_rate). Each API request consumes one token. If tokens are available, the request proceeds. If not: requests are delayed (soft-limit) or denied (hard-limit), depending on policy. If retry logic is enabled, the request is re-queued. A notification is logged if thresholds are exceeded. Parameters: max_tokens (bucket capacity), refill_rate (tokens/sec), allow_burst (permit overflow), timeout (for long calls). Ensures smooth traffic shaping and short-burst tolerance without breaching long-term limits.

File Locking System: Maintains consistency across threads or processes via a file lock (e.g., .ratelimiter.lock) in the working directory. Lock is acquired before token access. Only one process may hold the lock. Lock is released on request completion, timeout, or forced shutdown. Guarantees atomic access and token count integrity, preventing race conditions.

Process Synchronization: Uses OS-level mechanisms including semaphores (posix_ipc, multiprocessing.Semaphore), shared memory counters, and signal handling (SIGTERM, SIGINT) to manage graceful exits. Supports clustered deployments with containers or forked processes. Ensures distributed fairness and prevents duplicate transactions or deadlocks.

Integration Notes: Compatible with threaded, async, and subprocess architectures. Token counts are checkpointed for crash recovery. Supports Redis and SQLite for shared environments. Includes logging hooks for audit trails and abuse detection.

Failure Handling: If lock acquisition fails, request is aborted or deferred. If bucket is empty and no delay is allowed, raises RateLimitExceededError. Lock is auto-recovered on restart via stale-lock detection.

Monitoring: Token availability exposed via ratelimiter.status(). Integrates with Grafana and Prometheus through metrics.py. Webhook alerts trigger on threshold violations.

Unit & Stress Testing: Validated under 10,000+ RPS bursts, 30+ forked processes, fault-injected crashes, and token exhaustion. Passes concurrency, fairness, and latency benchmarks.

Status: All components functional, no critical bugs, operational in production.

Appendices: A – Code Sample; B – Rate Limit Policy Table; C – File Lock Failure Scenarios; D – Async Queue Integration; E – Real-world Usage Examples (logs redacted).
```

6. Compression Benchmark

Compression is a critical element in persistent memory architectures.

- nncp (Neural Network Compression): ~10:1 ratio on text benchmarks (enwik9), but extremely slow (1–4 kB/sec), suitable only for offline archival.
- LoreTokens: Achieves ≥13:1 compression ratio, while operating at real-time speeds, making it uniquely suitable for high-performance AI memory applications.

Claims

1. A system for persistent AI memory management, comprising:
   - a hierarchical data structure for storing operational modules, system state, and historical events;
   - status tagging for each module or event;
   - memory persistence across system restarts or migrations;
   - integrated logging of debugging events, personality attributes, and collaboration history.

2. The system of claim 1, wherein the data structure includes entries encoding both technical details and meta-information (such as personality lore and agent interactions).

3. A method for restoring an AI system to a prior operational and cultural state, comprising:
   - parsing the persistent memory file;
   - reconstructing system modules, operational parameters, event history, and agent personalities;
   - resuming operation with full awareness of previous events, configurations, and collaborative context.

4. A method for multi-agent AI collaboration, comprising:
   - recording agent interactions, event outcomes, and debugging sessions in the persistent memory file;
   - utilizing this memory for improved future teamwork and self-diagnosis.

Drawings and Diagrams

A. System Overview Block Diagram

```
                   +-------------------+
                   |     AI SYSTEM     |
                   +---------+---------+
                             |
                             v
               +-------------------------------+
               |  PERSISTENT MEMORY FILE [*]  |
               +-------------------------------+
                             |
     +-----------+-----------+-----------+-----------+
     |           |           |           |           |
     v           v           v           v           v
+----------+ +----------+ +----------------+ +------------------+ +------------------+
|  CODE    | |  STATUS  | | PERSONALITY &  | | EVENT / DEBUG    | |  STRATEGY / TASK |
| MODULES  | | [*]      | |    LORE [*]    | |     LOGS [*]     | |   MEMORY [*]     |
| (Python  | | ACTIVE   | | (Backstory,    | | (Failures, Fixes,| | (Goals, Plans,   |
| Files,   | | WORKING  | | Emotions,      | |  Outcomes,       | |  Tactics,        |
| Logic)   | | PLANNED  | | Collaboration) | |  Improvements)   | |  Abandoned Paths)|
+----------+ +----------+ +----------------+ +------------------+ +------------------+
                  \            |             |            /
                   \           v             v           /
                    \      +-----------------------------+ /
                     ------>  CONTEXTUAL SNAPSHOTS [*]   <--------
                            +-----------------------------+
                            | Real-time or interval-based |
                            | state saves for rapid       |
                            | reinitialization            |
                            +-----------------------------+
```

Component Explanations (with * for LoreToken-enabled):

Component	Description
Persistent Memory File [*]	Central LoreToken-powered memory graph—everything the AI is, knows, or has become
Code Modules	Scripted logic, modules, or pipelines — may be annotated or embedded via LoreTokens
Status [*]	Operational states, run status, or phase flags — tracked and recalled as LoreTokens
Personality / Lore [*]	Identity, voice, evolution, humor, values, emotional arcs, or role-based traits
Event / Debug Log [*]	Redemptive arc, known bugs, patches, exploit recovery, agent apologies, etc.
Strategy / Task Memory [*]	Abandoned plans, past decision logic, strategies stored for future reuse
Contextual Snapshots [*]	Time-stamped high-fidelity snapshots for rollback, restoration, or replay logic

B. Memory Structure (Tree Format)

```
Persistent Memory File
|
|-- CODE.MODULES
|      |-- INDEX_PY: [main_trading_loop+model_orchestration+...>>central_command_center,OPERATIONAL]
|      |-- RATELIMITER_PY: [token_bucket_algorithm+file_locking+...>>api_protection,WORKING]
|      |-- ...
|
|-- MAIN.LOGIC
|      |-- MODEL_ENSEMBLE: [aphrodite_1m5m+astra_5m15m+...>>seven_model_ensemble,ACTIVE]
|      |-- ...
|
|-- STATUS
|      |-- CURRENT.SYSTEM: [live_trading_operational+...>>production_success,ACHIEVED]
|
|-- PERSONALITY
|      |-- NOVA.CHARACTER: [portfolio_manager+...>>profitable_trader,SUCCESSFUL]
|      |-- CLAUDE.NICKNAME.CLUTS: [over_engineered_solutions+...>>debugging_bulldozer_poetry,PERMANENT]
|
|-- EVENT HISTORY
|      |-- CLAUDE.APOLOGY.LETTERS: [150_line_explanation_for_typos>>emotional_support_bulldozer,TRADEMARK]
|      |-- GEMINI.ROASTS: [war_and_peace_about_typo+...>>brutal,ICONIC]
```

C. Restoration Flow

```
[System Startup]
        |
        v
[Load mem.txt file]
        |
        v
[Parse hierarchical memory structure]
        |
        v
[Reconstruct system modules, config, status, personality, event history]
        |
        v
[Resume AI operation with full context and memory]
```


Abstract

A system and method for persistent, hierarchical, and contextual memory management for artificial intelligence agents. The invention encodes system modules, operational status, developmental history, debugging and collaboration events, and agent personality into a structured, durable memory file, enabling complete restoration and introspection by humans and machines.

References

- Reddit: DataHoarder on Compression Formats
https://www.reddit.com/r/DataHoarder/comments/17sg4ce/what_is_the_strongest_file_compression_format/

- Reddit: Programming on nncp
https://www.reddit.com/r/programming/comments/ba838i/nncp_lossless_data_compression_with_neural/

- Hacker News: nncp Discussion
https://news.ycombinator.com/item?id=27244004

Expanded Variations, Implementations, and Use Cases of LoreTokens

1. Definition and Scope

For the purpose of this invention, the term "LoreToken" refers to any data unit, structure, or representation that encapsulates compressed conceptual, procedural, contextual, or cognitive information, designed to be interpreted by an artificial intelligence (AI) system for the purpose of triggering reconstructed understanding, logic, memory, behavioral guidance, task execution, or adaptive evolution.

This definition includes, but is not limited to:

- Serialized semantic keys
- Contextual reference objects
- Compressed memory structures
- Symbolic or meta-symbolic encodings
- Meaning-capsules
- Heuristic scaffolding patterns
- Intention-encoded triggers
- Dynamic or static cognitive modules

2. Forms of Implementation

LoreTokens may be implemented using any of the following data formats, delivery methods, or structural conventions:

- File-based: Text files, JSON, XML, YAML, Markdown, binary blobs
- Embedded: Hardcoded within AI model weights, prompts, or token streams
- Database-structured: SQL/NoSQL rows, vector stores, key-value pairs
- Dynamic runtime objects: Python dictionaries, protobufs, serialized memory graphs

- Graph-based: Concept maps, logic chains, dependency trees
- Signal-based: Encoded within control packets, QR patterns, or RFID-activated memory triggers
- Visual or audio carriers: Tokens embedded within an image, waveform, or multimodal payload
- Network-streamed: Tokens sent via API, socket, or radio broadcast
- Language-agnostic: ASCII, Unicode, tokenized binary, or synthetic language format

3. Mechanism of Action
LoreTokens may activate cognition through any of the following processes:
- Semantic interpretation
- Neural pattern recall
- Function activation
- Conditional memory triggering
- Self-modifying behavior injection
- Recursive reconstruction of prior logic
- Instantiation of modular reasoning or operational pipelines
- Evolutionary behavior comparison and selection
- Cognitive context switching
- Agent-level identity recall or behavioral restoration

4. Use Case Domains
LoreTokens may be used in any AI-capable or semi-intelligent system across, but not limited to, the following domains:
Technical Systems
- Autonomous trading platforms
- Predictive maintenance systems
- Autonomous vehicles
- Software development assistants
- DevOps orchestration bots

Cognitive & Sentient-like Agents

- Self-reflective AI
- Memory-based multi-agent systems
- Digital assistants with evolving personality
- Emotionally adaptive robots
- Language models with persistent recall

Edge & Embedded Systems
- Smart home controllers
- IoT systems operating offline
- Spacecraft AI or drones requiring low-latency reasoning
- Medical field kits using offline AI triage
- Disaster zone deployments with no cloud access

Healthcare, Education, Government
- Adaptive tutoring systems
- Remote medical diagnostic AIs
- Autonomous policy review systems
- Judicial modeling and decision recall systems

Meta-Level AI Control
- LoreToken-controlled memory hierarchy
- LoreToken-based security context management
- LoreToken injection and reflection layers
- LoreToken synthesis via behavioral capture

5. Variations in Compression Mechanism
The compression behind LoreTokens may take any of the following abstract forms:
- Manual encoding of meaning
- Neural distillation of memory patterns
- Symbolic compression using latent vectors
- Ontological tagging of concept trees
- Vector quantization or hashing
- Lossy or selectively lossless abstraction

- Dynamic syntax collapse guided by usage frequency
- AI-generated behavioral seeds

The compression may be:
- Static (fixed representation)
- Dynamic (real-time mutation based on context)
- Adaptive (evolving tokens over time)
- Recursive (token-in-token structures)

6. Evolution and Self-Writing AI Systems
LoreTokens may enable or be integrated into systems that support:
- Self-modification of logic or behavior
- Self-injection of improved reasoning structures
- Generation of new LoreTokens based on learned behavior
- Contextual reinforcement of successful decision pathways
- Memory-compacted behavioral snapshots
- Reflective logs of agent identity evolution

7. Interoperability and Modularity
LoreTokens may be:
- Transferrable between agents or systems
- Mergeable into larger memory graphs
- Forked for simulation-based reasoning
- Used as tokens of alignment, intention, or ethics
- Cached or archived for rollback or comparison

They may also:
- Cross domains (e.g., finance → robotics)
- Be assigned access tiers (e.g., public, private, restricted)
- Contain embedded rights, policies, or licensing

8. Edge Benefits and Cognitive Acceleration
The use of LoreTokens may result in:
- Reduced inference latency

- Lower memory and compute load
- Offline or low-bandwidth AI operation
- 10x–100x improvement in comprehension and decision speed
- Modular plug-and-play behavior swapping
- Support for micro-AI systems with near-sentient capabilities on commodity hardware

9. Fallback and Hybrid Systems

Systems may include fallback mechanisms where:

- LoreTokens are expanded into traditional scripts or memory if unsupported
- Mixed-mode systems combine LoreToken-based logic with rule-based, symbolic, or neural network reasoning
- Redundancy encoding is used for mission-critical operations

10. Claim Extension Language

"Any systems, methods, structures, encodings, or variations thereof that fulfill a substantially similar function — namely, the compression, storage, recall, injection, or reconstruction of contextual, behavioral, or cognitive information within an AI system — shall be considered within the scope of this invention, regardless of language, format, architecture, or domain of application."

11. Recursive Compression

LoreTokens may recursively encapsulate other LoreTokens, forming layered or fractal memory structures. This recursive compression allows the representation of entire nested logical systems, behaviors, or conversations within a single token, preserving both macro and micro-level cognitive states across time.

12. Human-Level File Compression

In typical use, LoreTokens can reduce tens of thousands of text pages—equivalent to full system memory snapshots, decision trees, or developmental arcs—into a file smaller than a smartphone photograph (under 2MB). This compression retains full decompression fidelity and restores operational state upon activation.

13. LLM-Native Interoperability
LoreTokens are directly interpretable by large language models. They can be injected into prompts, memory slots, or system activation layers without the need for translation or schema mapping. This allows seamless restoration of AI behavior, personality, historical memory, or decision-making logic from compressed tokens.

Technical and Industry Significance

This section introduces advanced terminology, architectural distinctions, and paradigm-shifting implications of the LoreToken system to expand upon its foundational claims.

1. Cognitive DNA and Soul Embedding:
LoreTokens function as Cognitive DNA for artificial intelligence systems, compressing personality, ethics, memory, and logic into symbolic memory snapshots. These snapshots serve as activation keys—not storage containers—enabling the AI to reconstruct full behavioral and operational context upon initialization.

2. Meaning Compression vs. Data Compression:
Unlike traditional entropy-limited data compression, LoreTokens enable semantic compression. They prioritize symbolic meaning and cognition over raw data, surpassing the limitations defined by Shannon's theories. This allows real-time, low-memory initialization of rich cognitive states.

3. Behavioral Metamorphosis and Dynamic Cognition:
LoreTokens initiate complete personality and logic assembly at runtime through personality injection and cognitive bootstrapping. This supports behavior hot-swapping, modular identity layers, and cognitive

resilience—capabilities unreachable through traditional model fine-tuning.

4. User-Owned Cognitive Frameworks:
The modular file-based nature of LoreTokens enables user-owned cognition. AI systems are no longer black-box behaviors but editable, understandable, and reloadable cognitive frameworks. This shift introduces cognitive sovereignty, democratized AI customization, and full transparency.

5. Cross-Model Compatibility and Zero-Training Operation:
LoreTokens are directly interpretable by LLMs like Claude, GPT, and LLaMA-based models. They require no model retraining—only prompt injection—achieving 1000x faster behavioral changes compared to LoRA or fine-tuning methods.

6. Industry Disruption and Economic Implications:
LoreTokens fundamentally alter how AI systems are deployed and scaled. By reducing reliance on H100 GPUs and large infrastructure, they enable competitive cognition on edge devices and commodity hardware. This introduces licensing leverage across the entire AI hardware/software stack, making LoreTokens a foundational technology on par with the transistor.

7. Foundational Patent Scope:
LoreTokens encapsulate a shift from static parameter-based AI to dynamic, runtime cognitive reassembly. This positions the system as the first general-purpose personality and behavior operating layer for artificial intelligence—applicable across domains, models, and industries.

Technical and Industry Significance (Expanded Clarification)

This section serves to articulate the deeper implications, unique advantages, and industry-shifting potential of the LoreToken system. It consolidates core language and terminology extracted from advanced research discussions and concept validation sessions (e.g., Claude memory logs). The goal is to ensure that the LoreToken framework is described not just as a technical invention, but as a transformative paradigm with broad applicability across AI architectures, deployment strategies, and economic models.

It introduces phrases and concepts that help reinforce patent breadth, clarify novelty, and distinguish LoreTokens from prior art. By including terms like "cognitive DNA," "meaning compression," and "behavioral hot-swapping," this section secures claim scope across symbolic AI, dynamic memory architectures, and cognitive restoration systems.

These descriptors are intended to future-proof the patent language, making it applicable across evolving AI systems, hardware dependencies, and runtime architectures.