KairosChain: Pure Agent Skills with Self-Amendment for Auditable AI Evolution

Masaomi Hatakeyama
Genomics on Blockchain
18 January 2026

Abstract

The evolution of AI agent capabilities remains fundamentally opaque. When an AI system improves, changes behavior, or potentially becomes dangerous, the causal process of its capability formation cannot be verified by third parties. This paper introduces KairosChain, a meta ledger that records the evolution of AI skills on a private blockchain. KairosChain combines Pure Agent Skills—executable skill definitions using Ruby DSL and Abstract Syntax Trees (AST)—with the Minimum-Nomic principle, where rules can be amended but the amendment history cannot be erased. The system implements a layered architecture inspired by legal systems, separating immutable philosophy (constitution), evolvable meta-rules (law), project knowledge (ordinance), and temporary context (directive). By recording every skill state transition (who changed what, when, and how) on an immutable chain, KairosChain provides a novel approach to AI explainability: explaining not whether a result is correct, but how the intelligence that produced it was formed.

Keywords: AI explainability, agent skills, self-amendment, blockchain, audit trail, Minimum-Nomic, Ruby DSL, layered architecture, self-referential constraints

---

1. Introduction

Large Language Model (LLM) agents and AI coding assistants have become increasingly capable, yet their capability formation process remains a black box. Prompts are volatile, tool call histories are fragmented, and the evolution of skills—their redefinition, synthesis, and deletion—leaves no trace. As a result, even when AI agents become more capable, change their behavior, or exhibit potentially harmful properties, the causal process leading to their current state cannot be independently verified.

Current approaches to AI skill management, such as skills.md files, treat skills as static text documents. This approach suffers from fundamental limitations: skill interdependencies remain unverified, contradictions between skills go undetected, and the rationale for changes becomes untraceable. This situation resembles the anti-pattern of unlimited global variables in software engineering—initially convenient, but increasingly chaotic as the system grows.

This opacity poses significant challenges for AI governance and accountability [1]. Frameworks for evaluating AI skill composition have been proposed [2], yet these approaches focus on static snapshots rather than the dynamic evolution of capabilities over time.

KairosChain addresses this gap by providing an auditable record of AI skill evolution. Drawing on the concept of Minimum Nomic—a self-amendment game where rules can change but the change history is permanent [3]—KairosChain creates a meta ledger that answers the question: "How was this intelligence formed?"

The core insight of KairosChain is the explicit separation of three concepts:

- **STATE**: What capabilities currently exist
- **CHANGE**: How capabilities are being modified
- **CONSTRAINT**: Whether that modification is permitted

By treating change itself as a first-class citizen, KairosChain enables auditable evolution of AI capabilities.

---

2. Related Work

2.1 Agent Skills and Capability Modularization

Anthropic's Claude Agent Skills [4] introduced a paradigm for reusable, domain-specific expertise modules. Skills consist of metadata (always loaded), instructions (loaded on task invocation), and resources (loaded on demand), enabling efficient context window utilization while maintaining specialization. This three-layer architecture influences KairosChain's Pure Agent Skills design.

Yu et al. [2] proposed Skill-Mix, a framework for evaluating how AI models combine multiple skills to address complex tasks. While such frameworks assess skill composition at inference time, KairosChain focuses on making skill definitions themselves auditable and versioned over time.

2.2 Self-Amendment and Rule Dynamics

Hatakeyama and Hashimoto [3] proposed Minimum Nomic as a model for studying rule dynamics. Derived from Suber's original Nomic game [5], Minimum Nomic preserves the essence of self-amendment while promoting evolvability. The key insight is that rules can change, but the change process itself must be observable and recordable. KairosChain applies this principle to AI skill definitions: skills may evolve, but the evolution history is immutable.

2.3 AI Governance and Accountability

Priyanshu et al. [1] analyzed AI governance challenges using Claude as a case study, highlighting the need for transparency and accountability mechanisms. As AI agents become more autonomous, tracking how their objectives and capabilities evolve over time becomes essential. KairosChain provides infrastructure for such tracking at the skill definition level.

2.4 Domain-Specific Languages for Self-Describing Systems

Fowler [6] established that Ruby excels as a host language for internal DSLs due to its flexible syntax, block structures, and metaprogramming capabilities. This foundation enables KairosChain to create skill definitions that are simultaneously human-readable specifications and machine-executable code—a critical requirement for self-referential systems.

---

3. System Design

3.1 Why Ruby DSL: Language Choice for Self-Amendment

KairosChain's choice of Ruby DSL for skill definitions is deliberate and grounded in the unique requirements of self-amending systems. A self-referential skill system must satisfy three constraints simultaneously:

1. **Static analyzability**: Skills must be parseable without execution for security verification
2. **Runtime modifiability**: The system must be able to add, modify, or constrain skills during operation
3. **Human readability**: Skill definitions should serve as specifications readable by domain experts

Ruby uniquely satisfies all three constraints through its combination of features:

Expressive DSL Syntax. Ruby's block syntax (`do...end`), optional parentheses, and flexible method definitions enable skill definitions that read like natural language specifications:

```ruby
skill :core_safety do
  version "1.0"
  guarantees { immutable; always_enforced }
  evolve { deny :all }
  content <<~MD
    ## Core Safety Invariants
    1. Evolution requires explicit enablement
    2. Human approval required by default
  MD
end
```

The equivalent in Python or JavaScript would require significantly more syntactic scaffolding, reducing readability and increasing the gap between specification and implementation.

Static AST with Dynamic Introspection. Ruby provides `Ripper` and `RubyVM::AbstractSyntaxTree` in the standard library, enabling static parsing of skill definitions without execution. This allows security verification (checking for dangerous constructs) before any code runs. Simultaneously, Ruby's `define_method`, `class_eval`, and open classes enable runtime modification of skills—essential for the evolution capability.

Standard Library Completeness. Unlike JavaScript (requiring Babel/Acorn) or Python (limited `ast` module), Ruby's AST manipulation requires no external dependencies. Ruby 3.4+ further standardizes this with the Prism parser. This reduces attack surface and deployment complexity for security-critical systems.

Proven DSL Ecosystem. Ruby's DSL design patterns have been battle-tested for over two decades in Rails (web framework DSL), RSpec (testing DSL), Rake (build DSL), and Sinatra (routing DSL). These demonstrate that Ruby DSLs scale to production systems with millions of users.

For self-amending AI systems, Ruby thus represents a pragmatic optimum: expressive enough for readable skill specifications, powerful enough for runtime evolution, and analyzable enough for security verification.

3.2 The Minimum-Nomic Principle

KairosChain implements what we term the Minimum-Nomic principle for AI systems:

- **Amendable rules**: Skills (capabilities, behaviors, constraints) can be modified
- **Immutable history**: Who changed what, when, and how is permanently recorded
- **Controlled evolution**: Changes require explicit enablement and, by default, human approval

This avoids two extremes: completely fixed rules (no adaptation) and unrestricted self-modification (chaos). The result is an evolvable but auditable system.

The principle extends to self-referential constraint: **skill modification is constrained by skills themselves**. This creates an intentional bootstrap problem—to change how evolution works, one must follow the current evolution rules. The rules protect themselves from unauthorized modification while remaining amendable through proper channels.

3.3 Pure Agent Skills

KairosChain defines skills not as documentation but as executable structures using Ruby DSL. The term "Pure" draws an analogy to pure functions in functional programming: skills that constrain their own modification through explicit rules, minimize side effects, and maintain referential transparency. In this sense, Pure Agent Skills form an *Evolvable Internal Language*—a self-describing, self-constraining system of capability definitions.

Each skill definition includes:

- **Version**: Semantic versioning for change tracking
- **Guarantees**: Invariants the skill maintains (e.g., `immutable`, `always_enforced`)
- **Evolve rules**: What aspects of the skill can be modified and by whom
- **Behavior**: Executable logic (optional, for introspection capabilities)
- **Content**: Human-readable documentation and instructions

Each skill definition is parsed into an Abstract Syntax Tree (AST), enabling semantic differencing. When a skill changes, KairosChain computes the AST diff and records the hash of both the previous and new states.

3.4 Layered Skills Architecture

KairosChain implements a legal-system-inspired layered architecture for knowledge management. This design recognizes that not all knowledge requires the same level of constraint or auditability:

Layer	Legal Analogy	Path	Blockchain Record	Mutability
L0-A	Constitution	skills/ kairos.md	None	Read-only (human-managed)
L0-B	Law	skills/ kairos.rb	Full transaction	Human approval required
L1	Ordinance	knowledge/	Hash reference only	Lightweight constraints
L2	Directive	context/	None	Free modification

L0: Kairos Core. The foundation layer contains meta-rules about self-modification. `kairos.md` embodies the system's philosophy and principles—immutable and managed outside the system through human consensus. `kairos.rb` contains executable meta-skills that govern evolution:

- `core_safety`: Immutable safety invariants (`evolve deny :all`)
- `evolution_rules`: Constraints on how skills can evolve

- `layer_awareness`: Understanding of the layer architecture
- `approval_workflow`: Human approval process definition
- `self_inspection`: Ability to examine own capabilities
- `chain_awareness`: Understanding of blockchain state

L1: Knowledge Layer. Project-specific universal knowledge in Anthropic Skills format (YAML frontmatter + Markdown). Changes are tracked with hash references on the blockchain but do not require human approval. Suitable for coding conventions, architecture documentation, and domain knowledge.

L2: Context Layer. Temporary context for AI working hypotheses and session notes. No blockchain recording, enabling free exploration. This layer provides AI agents the freedom to experiment without permanent records, while L0 and L1 maintain accountability for decisions that persist.

3.5 Blockchain as Meta Ledger

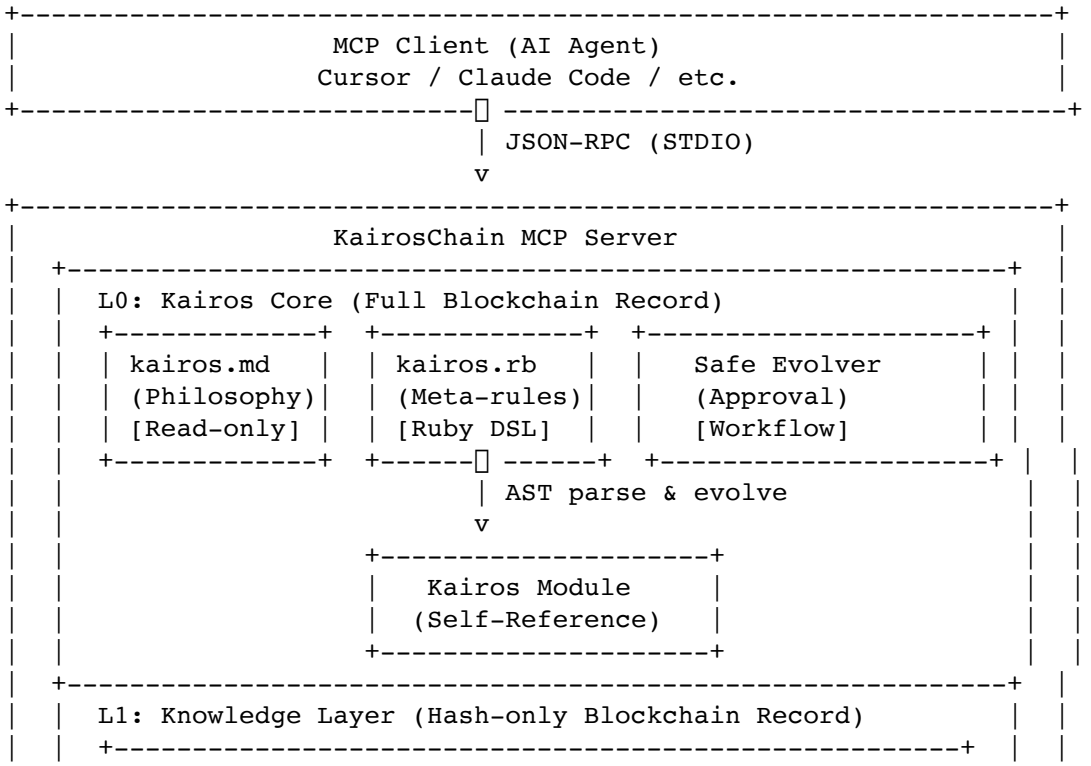
The minimal on-chain data structure is the SkillStateTransition:

Field	Description
skill_id	Skill identifier
prev_ast_hash	SHA-256 hash of previous AST
next_ast_hash	SHA-256 hash of new AST
diff_hash	SHA-256 hash of the AST diff
actor	"Human" / "AI" / "System"
agent_id	Agent identifier
timestamp	ISO 8601 timestamp
reason_ref	Off-chain reference to change rationale

AST content and detailed diffs are stored off-chain; only hashes appear on-chain, balancing auditability with storage efficiency.

---

4. Architecture



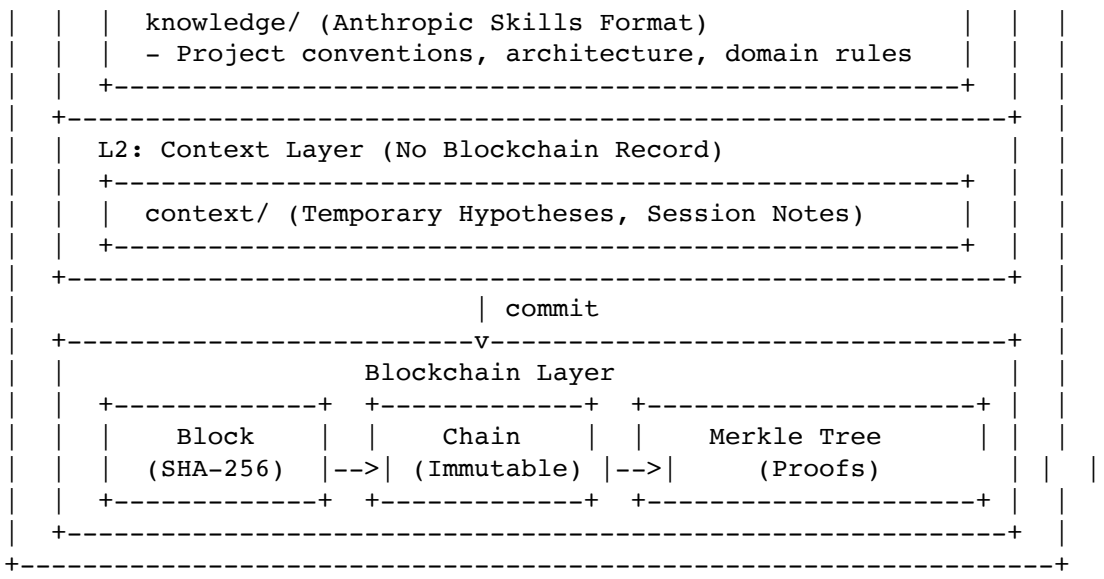


Figure 1: KairosChain Layered System Architecture

The architecture consists of three primary concerns:

1. **Layered Skills Management**: L0 (Kairos Core) manages meta-rules with full blockchain recording and human approval. L1 (Knowledge) stores project knowledge with hash-only recording. L2 (Context) provides free workspace for AI exploration.
2. **Self-Reference via Kairos Module**: AI agents can inspect their own skill definitions through the Kairos module, enabling introspective capabilities such as listing available skills, checking evolution permissions, and understanding constraints.
3. **Blockchain Layer**: Records skill state transitions immutably. Each block contains a Merkle root of transactions, enabling efficient verification of historical states.

KairosChain is implemented as a Model Context Protocol (MCP) server, allowing seamless integration with MCP-compatible AI agents such as Claude Code and Cursor.

---

5. Discussion

5.1 Transparency and Evolvability

KairosChain demonstrates that transparency and evolvability need not be mutually exclusive. By recording the history of changes rather than preventing changes, the system allows AI capabilities to grow while maintaining auditability. This aligns with the Minimum-Nomic principle: the game (AI operation) can evolve, but its evolution is always observable.

5.2 Limitations

Current limitations include: (1) reliance on off-chain storage for detailed AST data, (2) single-node operation in the initial implementation, and (3) the assumption that skill definitions adequately capture agent capabilities. The layered architecture partially addresses the third limitation by providing appropriate abstraction levels for different types of knowledge.

5.3 Team Operation and Governance Complexity

The current implementation assumes a single-approver model for skill evolution. When KairosChain is deployed in team environments, additional governance mechanisms become necessary:

Voting Systems. For L0 changes affecting the entire team, a multi-signature or voting mechanism may be required. This introduces complexity:

- Quorum requirements (minimum participation threshold)
- Approval thresholds (simple majority, supermajority, unanimity)

- Voting periods and deadlines
- Veto rights for critical changes

Recommended Approaches by Team Size:

Team Size	Recommended Approach
Individual	Current single-approver model
Small (2-5)	Git-based sharing with informal consensus
Medium (5-20)	HTTP API with voting tools (`governance_propose`, `governance_vote`)
Large (20+)	Full DAO-style governance with on-chain voting

kairos.md Special Consideration. As the "constitution" layer, changes to `kairos.md` should occur outside the automated system through human deliberation (e.g., GitHub Discussions, team meetings). This ensures that foundational philosophy changes receive appropriate scrutiny rather than being subject to automated workflows.

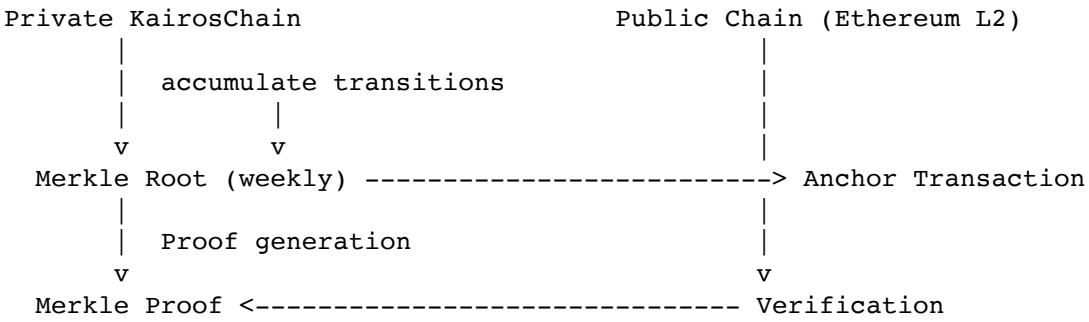
5.4 Public Blockchain Extension Considerations

While KairosChain currently operates as a private blockchain, extension to public chains introduces several considerations:

Gas Costs. Recording every skill transition on Ethereum mainnet would be prohibitively expensive. The recommended pattern is *Periodic Hash Anchoring*: accumulate transitions in the private chain and periodically commit only the Merkle root to the public chain. This provides public verifiability without per-transaction costs.

Privacy Concerns. Enterprise skill definitions may contain proprietary knowledge. Full publication on public chains creates intellectual property risks. Hash-only anchoring preserves verifiability (proving a state existed at a given time) without revealing content.

Recommended Architecture for Public Extension:



Layer 2 Solutions. For applications requiring more frequent public anchoring, Ethereum L2 solutions (Optimism, Arbitrum, Base) or dedicated rollups significantly reduce costs while maintaining security guarantees.

5.5 Experimental Observation: Self-Referential Improvement

To validate KairosChain's self-amendment capability, we conducted an experiment where KairosChain was used to improve itself. The AI agent (Cursor with KairosChain MCP enabled) was instructed to analyze its own codebase, extract coding conventions, and persist them as L1 knowledge.

Experimental Setup: - Environment: Cursor IDE (2.3.41) with KairosChain MCP Server - Date: 18 January 2026 - Task: Extract Ruby style conventions from `lib/kairos_mcp/` and save to L1

Procedure: 1. The AI analyzed actual code in `lib/kairos_mcp/` 2. Extracted naming conventions, error handling patterns, and Ruby idioms 3. Created `ruby_style_guide` in L1 using `knowledge_update` 4. Verified the creation using `knowledge_get` 5. Confirmed blockchain recording using `chain_history`

Results:

The following blockchain record was created:

```
Block #1
+-- Type: knowledge_update
+-- Layer: L1
+-- Knowledge ID: ruby_style_guide
+-- Action: create
+-- Content Hash: ae04bf58...
+-- Reason: "KairosChain self-improvement: Extract Ruby style
|           conventions from actual codebase"
+-- Timestamp: 2026-01-18T11:51:24+01:00
```

Interpretation:

This experiment demonstrates several key properties of KairosChain:

1. **Self-Referential Capability**: The system successfully analyzed its own code and extracted meaningful patterns—a form of self-inspection that is foundational to self-improvement.
2. **L1 Layer Functioning**: Knowledge was persisted in the appropriate layer (L1) with hash-only blockchain recording, as designed. The content hash `ae04bf58...` provides verifiability without storing full content on-chain.
3. **Immutable Audit Trail**: The reason field ("KairosChain self-improvement…") and timestamp create an auditable record of why and when the change occurred.
4. **Meta-Demonstration**: This is not merely documentation generation—it is a system recording knowledge about improving itself within its own infrastructure. The paper you are reading was itself improved using this workflow.

The full demonstration log is available at: cursor_kairoschain_demo_log_20260118_en.md

Source code: https://github.com/masaomi/KairosChain_2026

5.6 Future Directions

Planned extensions include:

1. **Ethereum Hash Anchoring**: Periodic anchoring to public chains for external verifiability
2. **Multi-Agent Federation**: Track multiple AI agents via `agent_id` with cross-agent skill sharing
3. **Zero-Knowledge Proofs**: Privacy-preserving verification of skill states
4. **Web Dashboard**: Visualization of skill evolution history and governance decisions
5. **Governance Tools**: Voting and proposal mechanisms for team deployment

---

6. Conclusion

KairosChain provides a novel approach to AI explainability by focusing not on whether AI outputs are correct, but on how the AI's capabilities were formed. By combining Pure Agent Skills (executable Ruby DSL definitions) with the Minimum-Nomic principle (amendable rules with immutable history), KairosChain creates an auditable trail of AI skill evolution.

The layered architecture—separating constitution, law, ordinance, and directive—enables appropriate constraint levels for different types of knowledge. This legal-system-inspired design recognizes that not all knowledge requires the same level of scrutiny: temporary hypotheses should be freely explorable, while core safety rules must be immutable.

The choice of Ruby DSL as the skill definition language is not incidental but essential. Ruby's unique combination of expressive syntax, standard-library AST tools, and runtime metaprogramming capabilities makes it ideally suited for self-amending systems that must be simultaneously human-readable, statically analyzable, and dynamically evolvable.

As AI systems become increasingly autonomous and capable, the ability to verify the causal process of their capability formation becomes essential. KairosChain offers a concrete step toward this goal: a meta ledger that answers the question, "How was this intelligence formed?"

Ultimately, KairosChain aims to provide a minimal yet verifiable institutional design for the co-evolution of humans and AI systems.

References

[1] A. Priyanshu, Y. Maurya, and Z. Hong, "AI Governance and Accountability: An Analysis of Anthropic's Claude," *arXiv preprint arXiv:2407.01557*, 2024.

[2] D. Yu, S. Kaur, A. Gupta, J. Brown-Cohen, A. Goyal, and S. Arora, "Skill-Mix: A Flexible and Expandable Family of Evaluations for AI Models," *arXiv preprint arXiv:2310.17567*, 2023.

[3] M. Hatakeyama and T. Hashimoto, "Minimum Nomic: A Tool for Studying Rule Dynamics," *Artificial Life and Robotics*, vol. 13, no. 2, pp. 500–503, 2009. DOI: 10.1007/s10015-008-0605-6

[4] Anthropic, "Agent Skills," *Claude Documentation*, 2025. [Online]. Available: https://docs.anthropic.com/en/docs/agents-and-tools/agent-skills

[5] P. Suber, *The Paradox of Self-Amendment: A Study of Logic, Law, Omnipotence, and Change*. New York: Peter Lang, 1990. ISBN: 978-0820412122

[6] M. Fowler, *Domain-Specific Languages*. Boston: Addison-Wesley, 2010. ISBN: 978-0321712943

DOI and Citation

This paper is available on Zenodo.

Recommended citation:

Hatakeyama, M. (2026). KairosChain: Pure Agent Skills with Self-Amendment for Auditable AI Evolution. Version 2.1. Zenodo. https://doi.org/10.5281/zenodo.18289164

Version 2.1 — 18 January 2026