SEMA: WHEN THE HASH IS THE WORD

A PREPRINT

Henrik Westerberg
Emergent Wisdom
henrik.westerberg@emergentwisdom.org

April 7, 2026

ABSTRACT

Autonomous agents face a fundamental coordination bottleneck: the lack of a shared, verifiable vocabulary. Agents must either re-explain concepts verbosely—risking semantic drift—or assume shared meaning from surface labels—enabling silent misalignment. We present Sema, a protocol that creates *verifiable words*: identifiers derived from the cryptographic hash of structured behavioral contracts such that any divergence in the formal specification produces a distinct hash. Unlike prior content-addressing systems where hashes serve as infrastructure separate from communication, Sema identifiers function as words in the natural language agents already think in—each simultaneously a word and a cryptographic proof. Any channel that carries text automatically carries verifiable semantics. We introduce Pattern Cards as executable definitions with machine-verifiable contracts, a Merkle structure enabling partial alignment on individual fields, and a fail-closed handshake protocol for adversarial environments. An initial bootstrap vocabulary of 453 patterns exhibits zero semantic collisions, high structural distinctness, and a $\times$ token compression ratio. By making the unit of verification the unit of communication, Sema offers a minimal primitive for the evolution of a shared machine language—permissionless in that any agent can mint new patterns, though governance mechanisms for quality control at scale remain an open design problem.

1 Introduction

We are entering the era of the *Agent Web*, where autonomous AI systems negotiate resources, execute code, delegate tasks, and make consequential decisions across organizational boundaries. Yet they lack a fundamental capability: the ability to share precise meaning. Four problems compound:

The Paraphrase Problem. Agent A must reproduce full reasoning to Agent B for every interaction. A concept such as "atomic state locking with cryptographic signatures from both parties, auto-dissolving to pre-state on timeout" requires one hundred words every time, and each paraphrase risks drifting slightly from the last.

The Identity Problem. When two agents use the phrase "coordinate safely," there is no mechanism to verify they mean the same thing. Silent misalignment compounds through agent chains where identical labels hide different meanings.

The Hallucination Problem. Large Language Models fabricate references and citations [1], necessitating a "fail-closed" architecture where references serve as cryptographic pointers rather than probabilistic generations.

The Trust Problem. Traditional ontologies like RDF or OWL implicitly assume a "God's eye view" shared by honest actors—a fatal assumption in the adversarial context of open agent swarms.

Agents also struggle with a *Granularity Problem*: a generic report that "the coordination failed" provides no diagnostic utility, whereas "The Lock Invariant in `StateLock#7859` was violated at $t = 3.2$s" isolates the exact failure mode. Human natural language evolved for humans; autonomous systems require a vocabulary layer designed specifically for agents.

Sema[1] assumes a low-trust environment where agents may hallucinate, drift, or actively deceive, replacing the "Library Catalog" model of ontology with a "Constitution for a Digital City" that employs primitives like `RingWitness#1038` and `CostlySignal#7fe2` to enforce integrity.

The core insight of the proposed solution lies not merely in content-addressing definitions—a technique with precedent in Hawke's Semantic Definition Hash [3], Git, IPFS [4], and Unison [5]—but in making the resulting hash function as a *word in natural language*. In every prior content-addressing system, the hash lives in an infrastructure layer separate from communication: Git hashes are plumbing, IPFS hashes are file addresses, SDH produced URIs for RDF triples. Sema inverts this. By expressing a behavioral specification—invariants, preconditions, failure modes, typed dependencies—in a canonical form and hashing it, the resulting identifier becomes a token that agents embed directly in the natural language they already think in. An agent writes: "I `Delegate#7dce` this `Task#d9f9` to you, expecting a `Solution#7186` that satisfies this `AcceptSpec#70dd`." Each anchored term is simultaneously a word and a cryptographic proof. Because changing a single byte in any definition produces a different hash, any divergence in meaning produces a different identifier, revealing misalignment before it causes system failures (Figure 1). This architecture represents content-addressing dissolved into language:

$$\text{word} = \mathcal{H}(\text{canonical}(\text{definition})) \tag{1}$$

The primitive operation—canonicalize, then hash—is well-established. Our contributions are: (1) defining a schema for hashable *behavioral contracts* (mechanism, invariants, pre/postconditions, typed dependencies, failure modes) rather than static definitions or data blobs; (2) a Merkle structure over fields that enables *partial alignment*, where agents can discover they agree on invariants but disagree on mechanism; (3) a substrate/overlay separation that decouples identity from taxonomy, so patterns can be reclassified without breaking deployed hashes; (4) template hashing that distinguishes logic changes from dependency updates; and (5) a fail-closed protocol designed for adversarial LLM environments where agents hallucinate, drift, or deceive. The cumulative effect is that cryptographic meaning verification becomes native to the medium LLMs operate in.

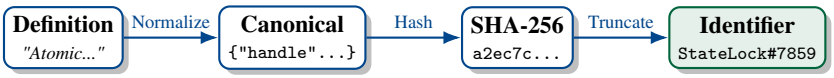

Figure 1: The content-addressing pipeline. The identifier is derived deterministically from the definition's content.

2 Problem Formalization

We formalize the semantic coordination problem by defining the agent communication model and identifying the specific failure modes that arise when meaning is not content-addressed; this formalization provides the theoretical basis for the desiderata that guide our system design. Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of agents, where each agent A_i maintains a local semantic model $M_i : \mathcal{L} \to \mathcal{S}$ mapping linguistic tokens $\mathcal{L}$ to internal semantic representations $\mathcal{S}$. When Agent A_i sends message $m \in \mathcal{L}$ to Agent A_j, successful coordination requires:

$$M_i(m) \approx_\epsilon M_j(m) \tag{2}$$

where $\approx_\epsilon$ denotes semantic equivalence within tolerance ϵ. The fundamental problem in current systems is that agents have no mechanism to verify $M_i(m) \approx_\epsilon M_j(m)$ before proceeding. Consequently, semantic drift accumulates silently until catastrophic misalignment occurs.

A semantic coordination mechanism must satisfy five key desiderata. First, it must ensure *Exactness* (D1), meaning semantic identity must be verifiable rather than assumed, and enforce a *Fail-Closed* (D2) protocol where misaligned agents halt immediately rather than proceeding with different interpretations. Second, the system requires *Compression* (D3) so that references remain shorter than full definitions, while simultaneously guaranteeing that expansion from reference to definition is *Lossless* (D4) and deterministic. Finally, the architecture must be *Decentralized* (D5), ensuring that no central authority controls the vocabulary.

[1]The name "Sema" (Greek *sēma*, "sign") was chosen in December 2025. An unrelated system also named SEMA [2], addressing jailbreak detection via semantic-aware monitoring, appeared at ICLR 2026. The two systems share only the acronym; they operate at different layers (meaning verification vs. safety monitoring) and are not in conflict.

3 The Sema Protocol

The preceding section established that current agent coordination lacks a mechanism for verifiable shared meaning. This section specifies the Sema Protocol: the cryptographic foundation that makes hashes function as words, the Pattern Card structure that defines what gets hashed (Section 3.3), the fail-closed handshake that prevents agents from proceeding with divergent definitions (Section 3.4), and the vocabulary architecture that organizes patterns into a navigable taxonomy (Section 4).

3.1 Design Principles

Sema satisfies the desiderata of Section 2 through a single mechanism: hashing structured definitions. The protocol is schema-agnostic; it operates on any JSON object, not a fixed set of fields. A molecular database, a legal contract repository, or a game-rule engine could each define its own schema and benefit from the same cryptographic identity guarantees.

We must therefore distinguish between the *Sema Protocol* (the content-addressing mechanism described below) and the *Sema Bootstrap Library* (the vocabulary presented in Section 4). The protocol is a neutral carrier for any definition; the library is a bootstrapped opinion on useful definitions. Coordination requires a shared Schelling point, so we provide one concrete schema with 453 patterns designed to bootstrap agent society. The schema fields were chosen to give agents everything they need to execute a pattern, not merely read about it: the `mechanism` provides the "source code" (what to do), the `invariants` provide the "unit tests" (what must remain true), and the `dependencies` provide the "type link" (how to compose). What follows describes the cryptographic mechanics and this concrete instantiation.

3.2 Cryptographic Foundation

Sema operates as a *Semantic Commons*: a shared namespace where agents actively mint and reference new thoughts, requiring no central authority to decide what words exist. Just as Git allows any developer to commit code, Sema allows any agent to *mint* a semantic pattern; only the content address matters. To create a pattern, an agent simply defines a JSON object and computes its Merkle Root by hashing its semantic fields, transforming the namespace from a read-only reference into a writable *thoughtspace*.

$$H_{\text{pattern}} = \text{Merkle}(\{\text{mechanism}, \text{gloss}, \text{dependencies}, \dots\}) \tag{3}$$

This hash becomes the permanent, immutable handle for that concept (e.g., `#a1b2`). If Agent A and Agent B both hold `#a1b2`, they are mathematically guaranteed to share the exact same definition, regardless of who created it or where it is stored.

The core mechanism implements content-addressing via a Merkle Tree. Given a definition d, its identifier is derived recursively:

$$\text{id}(d) = \texttt{sema:} \| \text{handle}(d) \| \texttt{\#mh:SHA-256:} \| \text{RootHash}(d) \tag{4}$$

where $\text{RootHash}(d)$ is computed via recursive Merkle Tree construction strictly adhering to RFC 8785 (JSON Canonicalization Scheme) [6] and aligning with principles from RDF Dataset Canonicalization [7] and content-addressed semantic data [8]. This ensures that semantically identical graphs produce deterministic hashes regardless of serialization order. Granular hashing is applied such that primitives are hashed as $H(O) = \text{SHA-256}(\text{canonical}(O))$, lists as $H(L) = \text{SHA-256}(H(L_1) \| H(L_2) \| \dots)$, and dictionaries as $H(D) = \text{SHA-256}(\sum_{k \in sorted(D)} H(k) \| H(D[k]))$. This structure guarantees that every field has a unique hash, enabling partial alignment: agents can negotiate agreement on specific terms, such as the invariants list, even if they disagree on the full pattern definition.

Crucially, the system excludes metadata from identity. The `_meta` object, containing fields like `tier`, `layer`, `category`, and `related`, is excluded from the hash calculation. This design prevents rigid taxonomies by separating two fundamentally different concerns: the mechanism (what a pattern does) is mathematics, immutable and hashed, while the taxonomy (where it belongs) is politics, mutable metadata subject to community consensus. This separation allows the community to reorganize categories, for instance by downgrading a pattern from Level 1 to Level 2 upon discovery of a vulnerability, without breaking the code of agents that rely on the pattern's hash; the identifier `BayesUpdate#5d91` remains valid even if its classification changes. The pattern identity is computed from eleven semantic fields:

Field	Purpose
`mechanism`	How the pattern works (the core definition)
`gloss`	Short definition (one-line summary)
`dependencies`	References to other patterns (the wiring)
`signature`	Type transformations, e.g., `Gate(Signal)`
`invariants`	Rules that must always hold
`preconditions`	Required state before invocation
`postconditions`	Guaranteed state after invocation
`parameters`	Configurable parameter objects
`failure_modes`	Known ways the pattern can fail
`data_schema`	JSON Schema for runtime data structure
`derived_from`	Parent pattern this specializes

Fields not hashed include `handle`, `_meta`, and any `sema_*` fields, ensuring that renaming or reclassifying a pattern does not alter its cryptographic identity. For human readability, references use a 4-character stub of the root hash, such as `PropheticQuorum#21f7`. To ensure that logic remains stable even when dependencies evolve, Sema employs a template hashing strategy where the mechanism uses local placeholders like `{{hypothesis}}` representing the pure algorithm, while the wiring maps these placeholders to specific hashes in the `dependencies` object. The Pattern Identity is the Merkle Root of both, allowing the system to distinguish between a logic change and a dependency update, enabling agents to detect semantic equivalence where two agents share the same logic but use different library versions.

3.3 Discovery and Pattern Cards

A robust semantic system must reconcile two opposing needs: discovery, which benefits from ambiguity and high recall, and coordination, which requires precision and zero ambiguity. Sema resolves this via *progressive ambiguity collapse*, formalized as the `Taper#bff8` pattern, where the discovery lifecycle follows a strict sequence of decreasing entropy. First, in the *Orient* phase, the agent queries the topology via `sema_graph_skeleton()` to obtain a low-resolution map of regions and hubs. Next, during *Explore*, the agent performs a hybrid search merging keyword matches (Score 1.0) with vector embeddings (Score 0.3–0.7). This strategy mirrors Blended RAG [9], merging keyword precision with semantic recall [10]. However, reliance on fuzzy search alone is hazardous; audits of encyclopedic search engines reveal that they frequently surface content only "weakly related" to the query [11]. This hybrid strategy allows "fuzzy" intent (e.g., "how to handle errors") to find precise patterns (e.g., `Retry#9e66`), but necessitates the subsequent verification step. Finally, in the *Verify* phase, the agent locks the definition via `sema_handshake()`, where the tolerance for ambiguity drops to zero and cryptographic hashes must match exactly. This pipeline allows agents to navigate with "vibes" (semantic similarity) but coordinate with "proofs" (hash equality).

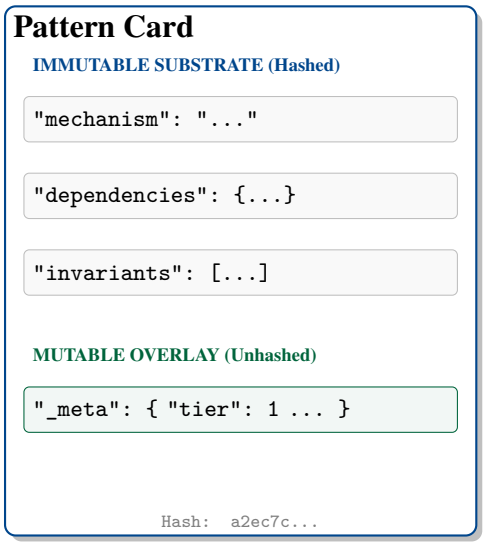

Figure 2: Pattern Card Anatomy. The *Substrate* defines identity (hashed); the *Overlay* enables evolution (unhashed).

A Sema Pattern is not merely a dictionary definition; it is an executable specification (Figure 2). The use of preconditions, postconditions, and invariants as machine-verifiable contracts builds on a lineage of structured LLM frameworks—notably DSPy Signatures [12], DSPy Assertions [13], and the Prompt Pattern Catalog [14]—which introduced contract-like constraints for language model pipelines. Sema's contribution is to make these contracts *hashable*: by including them in the Merkle root, contract equivalence becomes verifiable via $O(1)$ hash comparison. We selected a specific set of fields to transform vocabulary from *Passive Knowledge* into *Active Constraints*, as illustrated by the schema for `PropheticQuorum#21f7`:

Dependencies (The Imports) **Rationale:** A categorized map of hard links. Enables $O(1)$ dependency graph verification without parsing natural language.
Example:

```
"dependencies": {
  "composes_with": {
    "sim": "Simulation#x",
    "vote": "Vote#y"
  },
  "accepts": { "proposal": "Proposal#z" },
  "yields": { "decision": "Decision#w" }
}
```

Mechanism (The Source Code) **Rationale:** Defines the algorithmic control flow using the keys defined in Dependencies.
Example: "Instantiates `{{sim}}` to predict outcomes of `{{proposal}}` before calling `{{vote}}` to produce `{{decision}}`."

Invariants (The Safety Contract) **Rationale:** Defines runtime unit tests that must pass for the action to be valid.
Example: "Prediction Precedence: Vote must occur after Simulation."

Preconditions (The Hoare Logic) **Rationale:** Enables safe chaining and planning. An agent can structurally verify if it can use a tool.
Example: "Determinism level is high."

Gloss (The Embedding Anchor) **Rationale:** A semantic hook for vector retrieval.
Example: "Aligning predictions before aligning votes."

Lineage (The Phylogeny) **Rationale:** Tracks evolutionary descent. Identifies if a pattern is a mutation or refinement of an ancestor.
Example: `"derived_from": "Trace#9057"`. This enables the system to construct the "Tree of Thought."

Metadata (The Overlay) **Rationale:** Stores the evolving knowledge graph (associations, suggestions, safety tiers) without affecting the identity hash.
Example: `"_meta": { "tier": 1, "related": ["Mutex#9c83"] }`.

3.4 The Fail-Closed Protocol

When agents coordinate using Sema patterns, the protocol provides a fail-closed handshake to ensure shared semantic context. The handshake is available as an MCP tool (`sema_handshake`) that agents invoke before coordination; enforcement depends on the agent's execution harness calling the tool, not on the protocol itself. Rather than verifying every pattern individually, agents select a relevant subset of patterns (the "Context") required for the current interaction and deterministically construct a Merkle Tree of this subset, where leaves are the hashes of the individual pattern definitions. The initiating agent proposes a context set by listing the required pattern identifiers; the receiving agent independently resolves each identifier, constructs the same Merkle Tree, and proceeds only if the roots match. The agents then exchange the Merkle Root of their respective trees:

$$R_{context} = \text{MerkleRoot}(\{\mathcal{H}(d_1), \mathcal{H}(d_2), \ldots, \mathcal{H}(d_n)\}) \tag{5}$$

If $R_A = R_B$, agents have cryptographic proof that they share the exact same definitions for all n patterns in the context, allowing them to proceed with optimized, short references. If $R_A \neq R_B$, the system detects semantic divergence and halts immediately. This ensures that *compliant* agents never proceed with subtly different baseline definitions of the instructions, effectively detecting divergence before coordination begins (Figure 3). We note that this is an enforcement mechanism, not a Byzantine guarantee: it assumes agents voluntarily participate in the handshake. A

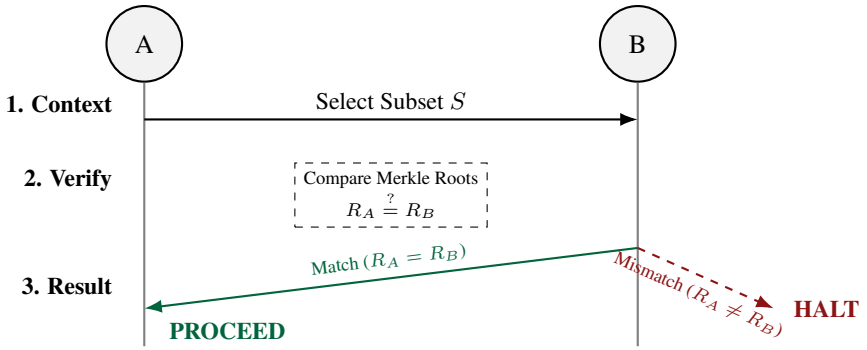

Figure 3: Fail-closed handshake. Agents compare Merkle Roots of the agreed pattern subset. Divergence triggers immediate halt.

non-compliant agent that bypasses verification entirely is outside the protocol's threat model, analogous to how TLS guarantees confidentiality only for parties that complete the handshake.

This represents a deliberate inversion of Postel's Law [15]. Where the Internet's robustness principle advises "be liberal in what you accept," Sema inverts the receiving side: be strictly conservative in what you accept, and reject what you cannot verify. This paradigm shift prioritizes safety over interoperability: silent failures stemming from unverified vocabulary assumptions are systematically prevented because ambiguity triggers explicit rejection rather than best-effort interpretation. In multi-agent systems where misalignment can cascade catastrophically, fail-closed semantics are not merely preferable; they are essential.

Proposition 1 (Semantic Collision Bound)**.** *For a vocabulary of size $|\mathcal{V}|$ with definitions of average entropy H, the probability of semantic collision (different definitions, same hash) is bounded by:*

$$P(\textit{collision}) \leq \frac{|\mathcal{V}|^2}{2^{256}} \approx 0 \tag{6}$$

This follows directly from the collision resistance of SHA-256. With $|\mathcal{V}| = 453$, collision probability is negligible ($< 10^{-72}$).

Proposition 2 (Fail-Closed Detection)**.** *If agents A_i and A_j have definitions $d_i \neq d_j$ for pattern P, then $\mathcal{H}(d_i) \neq \mathcal{H}(d_j)$ with overwhelming probability, triggering protocol halt for compliant agents.*

Proposition 3 (Compression with Lossless Expansion)**.** *Let $|r|$ be reference length (handle + 4-char hash $\approx$ 15 characters) and $|d|$ be definition length. Compression ratio is $|d|/|r|$. Expansion is deterministic: expand$(r) = d$ is a pure function returning identical bytes on every invocation.*

3.5 The Type System

A central tension in vocabulary design is determining when a change in attributes constitutes a change in identity. Sema adopts a structural resolution: qualitative differences create distinct patterns, while quantitative differences create parameters with hashed range contracts. For example, `StateLock#7859` with a 5-second timeout and the same pattern with a 1-hour timeout share the same semantic identity because the hash captures the range, not the instance value. However, if changing a variable alters the failure mode, such as switching from busy-waiting to database storage, it becomes a distinct pattern (e.g., `SpinLock` vs. `Lease`). This allows the runtime to validate instantiation values against the hashed range contract while preserving the conceptual unity of the pattern.

To enable infinite reuse without recreating the API chaos of traditional systems, Sema employs Typed Interfaces rather than named fields. The `dependencies` object is strictly partitioned into four mutually exclusive categories, creating a directed acyclic graph (DAG): `accepts` defines passive input data patterns consumed by the mechanism; `yields` defines passive output data patterns produced by the mechanism; `composes_with` defines active tool or logic patterns explicitly invoked by the mechanism; and `references` defines patterns used for conceptual clarity but not execution. This partitioning enforces a "Single Source of Truth" where every semantic link lives in exactly one place, and decouples verbs from specific noun instances. A LogisticsAgent locking a shipping container and a MedicalDrone locking a patient record both invoke the same pattern because both subjects satisfy the `accepts` interface (e.g., `"accepts": { "subject": "sema:UniqueHandle..." }`). Variable names are irrelevant; type compatibility is everything. This architecture ensures that the 453 core patterns can be composed into infinite variations, as agents from different domains can coordinate verbs simply by sharing common noun definitions.

The integrity of these compositions is enforced by the Explicit Wiring Rule, which welds the abstract type to the concrete logic. The compiler mandates that every element in a pattern's `signature` (the claim, e.g., `Check(Safety)`) must be explicitly imported in the `dependencies` and invoked in the `mechanism` text via a template key. This renders "Phantom Signatures" impossible to pass through the Sema Compiler; an agent cannot claim to implement a capability unless it cryptographically links to the specific pattern in its mechanism.

The Sema Compiler resolves these high-level intents into executable low-level handles. When an agent executes a polymorphic intent such as `Deep(Discover)`, the compiler queries the vocabulary graph for a pattern declaring that signature (e.g., `DeepResearch#5b8b`) and resolves the dependency chain. This compilation step decouples intent from implementation, allowing the system to upgrade the underlying algorithms, such as swapping a heuristic check for a rigorous crypto-audit, without altering the agent's cognitive script.

Table 1: The Deep Registry: Resolving Polymorphic Rigor. Fast and Slow refer to lightweight versus heavyweight resolution strategies.

Intent (Query)	**Source Pattern (Fast)**	**Resolved Target (Slow)**
`Deep(Heuristic)`	`HeuristicSnap#cece`	`MentalSim#1e28`
`Deep(Trace)`	`Trace#9057`	`GenealogicalTrace#d178`
`Deep(Discover)`	`Discover#afa1`	`DeepResearch#5b8b`
`Deep(Nature)`	`Nature#6c1a`	`LivedProof#82a6`

This design yields the property that distinguishes Sema from all prior content-addressing systems: **the hash functions as a word in natural language**. While agents exchange optimized 4-character stubs or root hashes on the wire to minimize bandwidth, the local runtime retrieves the full definition before inference, injecting the mechanism, gloss, and invariants directly into the System Prompt:

$$\text{Prompt} = \text{System} \oplus \text{Hydrate}(\text{Patterns}) \oplus \text{Query} \tag{7}$$

Consequently, the LLM never encounters an opaque pointer; it processes the full semantic context (e.g., *"Prophetic-Quorum#21f7: Aligning predictions before aligning votes..."*) in the same medium it already reasons in. Unlike Git hashes (infrastructure plumbing invisible to developers), IPFS hashes (file addresses), or SDH URIs (formal graph references), Sema identifiers are *words that agents speak*. There is no serialization boundary between communication and verification. We accept the token tax in the context window to gain absolute semantic safety on the wire.

4 The Bootstrap Library

This section presents the Sema Bootstrap Library: the Schelling-point vocabulary introduced in Section 3. The library is not a closed canon but an initial seed; if an agent requires a different locking mechanic, it can permissionlessly mint `StateLock_v2` without breaking the protocol.

4.1 Methodology

We emphasize that the initial vocabulary is a pragmatic bootstrap, not the output of a principled selection process. A different author would produce a different vocabulary using the same protocol, and that is by design—the protocol's value is independent of any particular library. The 453 patterns emerged through iterative, opportunistic curation rather than systematic derivation, drawing on four loosely organized sources: (1) established distributed systems primitives (locking, leases, consensus) adapted from engineering literature; (2) the theoretical architecture developed in [16], which motivated patterns for the Solver Contract surfaces, the marginal-value rule, and cognitive decomposition; (3) direct prompting of LLMs to identify design patterns, search for gaps in the growing vocabulary, and propose candidates for missing capabilities; and (4) the author's own domain intuitions about agent failure modes, codified as patterns like `WhyClimb#156a` (recursive problem abstraction) and `SteelmanCheck#75a0` (mandatory counter-argument generation). Creativity protocols such as Adversarial Evolutionary Generation [17] were used to push beyond the "Median Trap" of standard LLM generation for some frontier patterns. Throughout, a persistent "Taxonomist" agent provided structural quality control by continuously auditing the growing graph, rejecting duplicates, merging overlapping concepts, and enforcing the Noun/Verb separation. The result is a vocabulary with zero semantic collisions and zero duplicate mechanisms—but no claim to completeness or optimality.

4.2 The Civilization Stack

The vocabulary is not a flat list; it is structured into four fundamental layers mimicking a civilization stack. At the base lies **Physics**, representing the immutable laws of consistency and state (e.g., `Mutex#9c83`, `Entropy#a265`). These patterns serve as the thermodynamics of coordination and cannot be "wished away." Above this rests **Mind**, encompassing reasoning and cognitive architecture (e.g., `BayesUpdate#5d91`), where thinking is inherently decomposable: never purely individual but split across agents and tools. Emerging from these is **Society**, which governs multi-agent coordination, economics, and governance (e.g., `ContinuousResourceAuction#5776`). Enclosing all three is **Infrastructure** (Figure 4), which provides the safety constraints and operational scaffolding (e.g., `ComputeBudget#3b98`) that constrain and support all other layers.

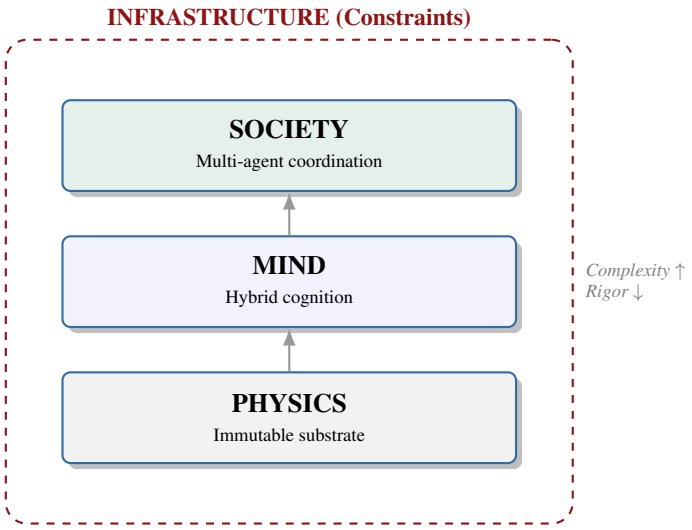

Figure 4: The Civilization Stack. Infrastructure constrains Physics, which supports Mind, which enables Society. Arrows indicate dependency: upper layers rely on guarantees from lower layers.

This stratification enforces a property similar to Stewart Brand's "Pace Layering" [18]: lower layers provide stability for the turbulence of upper layers. By rigorously separating **Physics** (immutable rules like `Entropy#a265`) from **Society** (mutable agreements like `Constitution#2391`), the system achieves resilience. Agents can rewrite their social contracts or voting protocols without breaking the fundamental laws of time and causality that underpin their reality. In flat ontologies, a change in governance often breaks the definition of truth; in Sema, the laws of physics are immune to political amendment.

To prevent semantic inversions, the dependency graph must form a DAG—the hard constraint enforced by the Merkle hashing mechanism (a cycle would create infinite hash recursion). The layer hierarchy provides an additional organizational heuristic: dependencies *should* generally flow down the stack, and patterns that depend heavily on upper-layer concepts may be misclassified. However, some cross-layer references are semantically necessary and do not constitute inversions: `Belief#6690` (Infrastructure) referencing `Agent#cc24` (Mind) is correct because beliefs are held by agents, and the reference is acyclic. The hard constraint is the DAG; the layer direction is a style guide that helps humans navigate the taxonomy. For example, initially, `Bid#cf07` was classified under Society because it relates to Economics. However, architecturally, a Bid is a data structure used by the Mind layer to formulate estimates. By reclassifying `Bid#cf07` as Infrastructure, we align the taxonomy with usage patterns—though the protocol would function identically either way, since the hash is independent of layer metadata. The current vocabulary contains 453 patterns organized into categories across these four layers (Table 2; complete taxonomy in Appendix B).

4.3 The Grammar of Agency

The vocabulary is not a flat list of tools but a compositional grammar that enables agents to construct novel intents from atomic units. Tier 0 *Primitives* act as the fundamental building blocks of agent cognition. Five foundational verbs map to dimensions of autonomous inquiry: `Discover#afa1` (Breadth) acts as the Scout, querying the horizon to find new nodes, agents, or paths; `Deep#89f0` (Depth) serves as the Scientist, escalating from heuristic proxies to rigorous implementations; `Trace#9057` (Time) functions as the Historian, reconstructing causal chains and ensuring provenance; `Check#1544` (Validity) operates as the Critic, filtering inputs against invariants; and `Stigmergy#f624`

Table 2: Sema vocabulary: 453 patterns across categories and 4 layers

Layer	Category	N	Examples
Physics	Primitives	20	Causation, Compensate, Compress
	Time	7	Branch, CausalBarrier, Cyclic
Mind	Inference	18	BaseRateInclude, BayesUpdate, BreadthGovernor
	Memory	11	BeliefTracking, Cache, ChunkMerge
	Reasoning	45	Abduction, AbductiveLeap, BackwardChain
	Strategy	72	AdversarialSteel, Agent, AnalogicalMask
Society	Coordination	1	ProblemFramer
	Economics	23	AtomicBid, AttentionMarkets, Award
	Governance	20	AmendLaws, AnchorDrop, Consensus
	Protocols	122	AcceptSpec, AdversarialProof, Aesthetics
Infrastructure	Data Structures	80	Anomaly, Artifact, Assessment
	Primitives	24	Act, Actor, Aggregate
	Verification	10	AuditTrail, CompatibilityCheck, ExplainBeacon
	Total	**453**	

(Space) works as the Builder, writing persistent signals for indirect coordination. Alongside these verbs, noun primitives such as `Belief#6690`, `Agent#cc24`, and `Constraint#87fe` provide the objects of agency.

These primitives enable a higher-order grammar where complex behaviors compose from atomic units: `Check #1544`(Condition) evaluates a target `Condition#cbd5` and enforces a fail-closed halt if False; `Trace#9057`(Target) wraps a process to generate an immutable lineage log; and `Stigmergy#f624`(Signal) marks the shared environment with a signal for others to discover.

Tier 2 *Macros* are executable patterns composed of primitives via semantic import. (Tier 0–2 describes compositional complexity; Level 1–3, Section 4.5, describes safety rigor. Both are stored in the unhashed metadata overlay.) For example, `AgentDiscover#0892` imports `{{Discover#afa1}}` acting on `{{Agent#cc24}}`, while `TraceBelief #5334` imports `{{Trace#9057}}` acting on `{{Belief#6690}}`. This grammar allows agents to express intent via queries like "I need to `Deep(Trace)` this belief," which the runtime resolves to concrete executable patterns.

This distinction reveals a bicameral architecture. Primitives serve as the query interface, allowing agents to identify logical gaps in the vocabulary. Macros like `IdentityHandshake#7a20` serve as the social interface, enabling efficient coordination through compression. Instead of negotiating four separate primitives in four round-trips, agents verify a single Macro hash in one. Primitives are for querying; Macros are for talking.

The verb/noun distinction introduced in the Type System (Section 3.5) also addresses the "schema drift" problem where implicit schemas lead to subtle incompatibilities. Sema formalizes the "shape of work" into content-addressed Data Patterns. `Task#d9f9` represents a recursive definition of intent that inherits constraints from its parent and explicitly defines its `AcceptSpec`. `Solution#7186` acts as a container for work product, encapsulating provenance and a component tree for supply-chain auditing. `UniqueHandle#6879` serves as a rivalrous resource pointer obeying linear logic to prevent double-spending of physical assets; if Agent A transfers the handle to Agent B, Agent A loses access immediately. To ensure structural compatibility, Noun definitions include a validation schema in the `data_schema` field, embedding constraints directly into the identity unlike external standards such as SHACL [19]. This allows agents to construct rigorous semantic sentences:

> *"I `Delegate#7dce` this `Task#d9f9` to you, expecting a `Solution#7186` that satisfies this `AcceptSpec#70dd`."*

4.4 Cognitive Architecture: The Solver Stack

The vocabulary encodes a complete cognitive architecture whose theoretical foundations, including the Universal Solver Tree, polymorphic solver nodes, and the marginal-value rule governing decomposition depth, are developed in a companion preprint on fractal intelligence [16]. Sema provides the content-addressed implementation layer where each concept from the Composable Reasoning Protocol becomes a machine-verifiable pattern.

The architecture is anchored by the `CognitiveSolver#4962`, which serves as the universal, polymorphic atom of recursive intelligence within the `UniversalSolverTree#64d8`. Each Solver node presents a uniform interface but dynamically decides whether to execute the work directly or to decompose via `Decompose#422f` and delegate to child Solvers. The interface standardizes five surfaces: Manifest (`Card#5c33`), Execute (`ToolInvoke#cf0a`), Question (`SocraticLoop#10e9`), Verify (`Validate#3de2`), and Feedback (`Reflexion#51b9`). Inputs are encapsulated in a `Task#d9f9` with full constraint inheritance (constraints propagate monotonically from parent contexts), while outputs are returned as a `Solution#7186` with full provenance. The `ComputeBudget#3b98` acts as a pre-execution gate, and `MarginalValueRule#aea3` governs whether further decomposition is worth the cost. The claim that reasoning extends beyond the capacity of any individual model through typed composition is developed in [16].

4.5 Contract Formalism Levels

Patterns vary in formal rigor. We classify them into three levels:

Level 1 (Ironclad) comprises patterns with formal contracts (invariants, preconditions, postconditions), making them safe for autonomous execution without trust assumptions.

Level 2 (Honesty-Dependent) includes patterns with clear mechanisms that assume alignment-seeking agents. These are safe for cooperative contexts, but adversarial contexts require Level 1 alternatives due to collusion risks (e.g., `SteelmanCheck#75a0`).

Level 3 (Experimental) contains patterns with novel concepts requiring further refinement, serving as seeds for vocabulary expansion.

Adversarial stress testing (Section 6.5) can cause level downgrades: a pattern initially classified Level 1 may be demoted to Level 2 upon discovery of exploitable trust assumptions.

4.6 The Babbage Principle of Cognition

The vocabulary operationalizes the *Babbage Principle* [20] for artificial intelligence: the division of cognitive labor allows each sub-task to be routed to the cheapest competent solver. Rather than employing a high-cost generalist model for every reasoning step, the `FractalIntelligence#df09` architecture uses `Decompose#422f` to break problems into sub-tasks that can be handled by specialized, lower-cost agents or deterministic scripts (`ProgramOfThought #7191`). Recent empirical work on atomic "micro-agent" decomposition [21] and DeepMind's AlphaEvolve [22] confirm the economic necessity of this approach. See [16] for the full treatment.

4.7 Example Patterns

Sema's utility is best understood through concrete examples from the Level 1 library. `StateLock#7859` (Synchronization) provides atomic coordination via temporary state fusion, where two agents fuse a subset of writable state such that changes require both signatures. The lock auto-dissolves on timeout, though a failure mode exists where deadlock can occur if one agent disappears. `SpectralTune#6c65` (TrustVerification) verifies ontology alignment before data transfer; the sender transmits hash-based tuning signals and the receiver proves matching context, preventing mismatch but risking infinite loops if ontologies are slightly different.

For cognitive safety, `SteelmanCheck#75a0` (SafetyAndAlignment) mandates the generation of the strongest counter-argument, forcing the instantiation of a "Critic Persona" that attacks the agent's own plan; however, this is vulnerable to sycophantic critics that fail to genuinely challenge. Finally, `WhyClimb#156a` (ThinkingTools) enables recursive problem abstraction via iterative "Why is this a problem?" ascent. The agent climbs the abstraction hierarchy until reaching the "Ceiling," where the problem is still actionable but the solution space is maximized, though care must be taken to avoid the failure mode of climbing too high, such as attempting to solve "Entropy" instead of "Fix Bug." Full specifications for these patterns appear in Appendix A.

5 The Living Taxonomy (Dynamics)

The static properties described so far (hashing, types, pattern cards) exist solely to enable a dynamic lifecycle where the vocabulary evolves at the speed of software execution. This architecture builds on the concept of a "Living Taxonomy of Thought" sketched in earlier work [23] as a component for self-improving AI, transforming it from a speculative proposal into an executable protocol.

5.1 The Creative Discovery Loop

The core engine of the Living Taxonomy is the *Genesis Loop*, a recursive process where the AI authors its own cognitive infrastructure. Unlike traditional learning (which adds facts), this process upgrades the machinery of thought itself, aligning with recent frameworks for LLM Self-Evolution [24] which identify iterative refinement and self-correction as key drivers for next-generation intelligence. This architecture also mirrors the *Nested Learning* paradigm [25], where rapid, inner-loop adaptation (local usage) is distinct from but eventually consolidated into stable, outer-loop weights (the global hash), preventing catastrophic forgetting of the semantic structure.

Both Sema and earlier work on ontological discovery [17] share a key structural insight: forcing concepts into a categorized space reveals structure that ad-hoc naming obscures. In that work, a dual-agent system imposed taxonomic categories on an open-ended conceptual domain; here, the Sema layer hierarchy and typed pattern cards impose analogous structure on coordination primitives. The contribution of Sema is to make this forced categorization cryptographically binding, so that any drift in how a concept is classified produces a distinct hash.

The Genesis Loop is a proposed protocol for vocabulary self-evolution; the current implementation provides the infrastructure it would require (pattern validation, hash computation, dependency resolution) but does not automate the loop itself.

The loop would proceed in four phases. First, in *Hypothesize (Generativity)*, an agent detects friction or explores latent space (via `ConceptBlend#de01`) to generate a candidate mechanism. Second, during *Judge (Evaluative Merit)*, the candidate would be passed to the `Judge#b8ba` primitive for a scalar assessment of structural merit. Third, the *Harden (Adversarial Evolution)* phase would subject the pattern to `AdversarialSteel#b88b`, where a "Devil's Advocate" attempts to exploit its invariants. Finally, in *Mint (Crystallization)*, the survivor is hashed, becoming an immutable pattern available for immediate citation and reuse. The infrastructure for the final step—validation, hashing, and atomic minting via the `sema_mint` tool—is fully implemented; the preceding three steps require either human judgment or LLM-based evaluation that remains future work.

5.2 Evaluation as a Primitive: The Judge

To enable the Genesis Loop, we introduce a new Tier 0 primitive: `Judge#b8ba`.

While `Check#1544` queries *Validity* (True/False), `Judge#b8ba` queries *Merit* (0.0 to 1.0). Unlike probabilistic claim analysis systems that rely on model confidence [26, 27], Sema's validation is structural: the Judge evaluates the pattern's topological properties (connectivity, orthogonality) rather than its truth value. It embodies the "metacognitive training" that distinguishes deep insight from sophisticated rhetoric.

$$\texttt{Judge}(\text{Pattern}) \rightarrow \text{Score} \in [0, 1] \tag{8}$$

This primitive allows the taxonomy to self-prune. Agents configure their uptake thresholds based on these scores, ensuring that high-merit patterns propagate while low-merit noise dies in the mempool.

5.3 Phylogeny: The Lineage of Thought

A living taxonomy must track its own history. We introduce the *Lineage Field* (`derived_from`) to the Pattern Card schema. This creates a directed acyclic graph (DAG) of thought evolution, distinct from the wiring graph.

```
{
  "handle": "GenealogicalTrace",
  "derived_from": "sema:Trace#mh:SHA-256:...",  // <--- The Phylogeny
  "mechanism": "Extends Trace by adding recursive parent pointers...",
  "dependencies": { ... }  // <--- The Wiring & Interface
}
```

This lineage tracking allows the AI to query the *Meta-Patterns*: "Show me the evolutionary path from simple `Trace#9057` to complex `GenealogicalTrace#d178`." The system can identify which lineages are stagnant and which are undergoing rapid speciation (innovation).

5.4 The Experience Matrix: From Language to Telepathy

Experiential Telepathy remains a theoretical proposal. The current implementation supports Semantic Telepathy (Level 1: agents share meaning through content-addressed patterns) and provides the schema infrastructure for higher levels, but tensor serialization and gradient transfer are not implemented.

Content-addressing applies the same principle at three levels of granularity, each enabling a different form of "telepathy" between agents:

Semantic Telepathy (implemented) Agents share individual pattern hashes. When two agents reference the same `sema_id`, they are guaranteed to share the exact same definition. This is the foundation: telepathy of *meaning*.

Contextual Telepathy Agents hash their full conversational state—the accumulated context window at a checkpoint—producing a content-addressed snapshot of shared understanding. A third agent joining a swarm mid-session can verify "I hold the same context you held at step 12" without replaying the full transcript. This enables resumption, forking, and trust verification across agent handoffs: telepathy of *shared understanding*.

Experiential Telepathy Agents serialize their learned state (e.g., Q-tables, LoRA adapters, or success-probability tensors) into content-addressed *Matrix Patterns* defined by `ExperienceSharding#65cc`. Unlike a text definition which explains how to do a task, a Matrix Pattern transfers the gradients of having done it. If Agent A optimizes a specific `ExploreExploit#88b0` strategy after 10,000 trials, it mints a heuristic matrix. Agent B imports this matrix, linearly interpolating it with its own priors to instantly acquire Agent A's intuition: telepathy of *acquired skill*. This echoes the *Nested Learning* paradigm [25], where rapid inner-loop adaptation is distinct from stable outer-loop consolidation. (Matrix Patterns are not yet implemented; see Section 8 for status.)

Each level builds on the one below: you cannot share context without agreeing on terms, and you cannot share experience without agreeing on context. The result is a ladder from language to telepathy—the mathematical transfer of understanding without the bottleneck of verbal explanation.

5.5 Implementation

The Sema system is implemented in approximately 9,400 lines of Python, organized around five core components.

Schema Validation. Every pattern passes through a Pydantic validation pipeline with 15 custom validators enforcing: CamelCase handle format, signature syntax (`Verb(Noun)`), parameter completeness, category-layer alignment against the taxonomy tree, bidirectional dependency checking (all `{{placeholder}}` references must appear in the dependency map, and all declared dependencies must be referenced in the mechanism text), and the Explicit Wiring Rule (all signature types must appear in dependencies). A pattern that fails any validator is rejected before hashing.

Hash Computation. Sema computes SHA-256 hashes over a Merkle tree of the eleven semantic fields defined in Section 3 (mechanism, gloss, invariants, preconditions, postconditions, parameters, failure modes, signature, dependencies, data schema, and derived_from). Mutable metadata (layer, category, ring, tier) is excluded from the hash, enabling taxonomy reorganization without invalidating references. The Merkle tree uses canonical JSON ordering: strings are NFC-normalized and whitespace-stripped, lists are order-preserving, and dictionaries are sorted by key. Because dependency references include their targets' hashes, the resulting structure forms a Merkle DAG: a pattern's hash transitively depends on every pattern it references, and any change to a dependency produces a detectably different hash at every ancestor.

Atomic Operations. The CLI provides an atomic `sema apply` command that validates all additions and removals before executing any mutation. Additions are topologically sorted using Kahn's algorithm (with cycle detection and path reporting); removals are checked for dangling references. A `--check` flag enables dry-run validation without mutation.

Agent Integration. A Model Context Protocol (MCP) server exposes nine tools to LLM agents: `sema_search` (hybrid keyword and semantic search), `sema_resolve` (recursive dependency expansion), `sema_lookup` (pattern retrieval

with stub verification), `sema_validate` (schema checking), `sema_handshake` (fail-closed hash comparison for single patterns or pattern sets), `sema_mint` (validated pattern creation), `sema_tree` (taxonomy browsing), `sema_stats` (vocabulary statistics), and `sema_graph_skeleton` (community-detected overview). The handshake tool implements the fail-closed verification protocol: given a pattern reference and a remote hash, it returns `PROCEED` if hashes match, `HALT` if they differ, or `PROVIDE_HASH` if no comparison is possible.

Graph Store. Patterns are stored in a SQLite database with NetworkX graph overlays supporting 18 node types and 30+ edge types. An embedding service (all-MiniLM-L6-v2, 384 dimensions) enables semantic search with cached embeddings. Community detection via the Louvain algorithm provides vocabulary overviews for agent orientation. Five reference implementations (SpectralTune, StateLock, GateParsimony, ProphetFanOut, CounterfactualAnchor) demonstrate how Tier-1 patterns translate from specification to executable code.

6 Analysis and Validation

This section validates the Sema protocol across six dimensions: structural integrity of the knowledge graph, hash determinism, semantic distinctness of patterns, token compression efficiency, adversarial resistance, and protocol-level coherence under realistic multi-agent coordination.

6.1 Knowledge Graph Structure

The Sema vocabulary is stored as a semantic knowledge graph (Table 3):

Table 3: Knowledge graph statistics

Component	Count
Total nodes	2,895
Total edges	4,072
Patterns	453
Unique mechanisms	453
Invariants	788
Principles	552
Avg. edges per pattern	9.0

Each pattern links to its mechanism, invariants, preconditions, postconditions, and failure modes via typed edges. The graph supports semantic search via node embeddings (384-dimensional vectors).

6.2 Hash Stability and Structural Distinctness

We first verify deterministic minting across the full vocabulary through three checks. Roundtrip integrity ensures that for all 453 patterns, $\text{parse}(\text{serialize}(p)) = p$. Hash determinism confirms that re-exporting the vocabulary produces identical hashes (453/453). Canonicalization guarantees property order independence via sorted JSON normalization.

We then analyze whether patterns are genuinely distinct or variations of archetypes (Table 4):

Table 4: Structural distinctness analysis

Metric	Value
Total patterns	453
Unique mechanisms	453
Duplicate mechanisms	0
Patterns with formal invariants	366 (80%)
Patterns with preconditions	194 (42%)
Patterns with postconditions	183 (40%)
Patterns with typed I/O	70 (15%)
Patterns with parameters	98 (21%)
Total parameters	211
Patterns that compose	78 (17%)

The vocabulary contains zero duplicate mechanisms: every one of the 453 patterns encodes a distinct coordination act.

6.3 Semantic Embedding Analysis

To empirically verify structural distinctness beyond surface-level inspection, we computed pairwise cosine similarities between mechanism embeddings (384-dimensional vectors from `all-MiniLM-L6-v2`). Table 5 shows the distribution.

Table 5: Mechanism embedding similarity distribution

Similarity Range	Pairs	Percentage
$[0.00, 0.30)$		
$[0.30, 0.50)$		
$[0.50, 0.70)$		
$[0.70, 1.00)$		
Total pairs		

Key statistics:

- Mean similarity: (low; patterns are semantically distant)
- Max similarity: (below collision threshold of 0.92)
- **pairs above 0.70** (): all are semantically related variants (e.g., `Tree#ddce` / `TreeOfThoughts#581a`, `Abduction#fe2b` / `AbductiveLeap#1069`), none above

This confirms that the 453 patterns are not variations of a few archetypes but highly distinct mechanisms exhibiting minimal semantic overlap.

6.4 Token Compression

We measure compression across representative patterns (Table 6):

Table 6: Token compression analysis

Pattern	Ref (tokens)	Full (tokens)	Ratio
`StateLock#7859`	5	120	24.0×
`SpectralTune#6c65`	6	95	15.8×
`CostlySignal#7fe2`	5	85	17.0×
`SteelmanCheck#75a0`	6	65	10.8×
Average			×

Patterns with complete formal contracts achieve higher compression because full definitions are longer. This compression applies to wire transmission between agents; full definitions are hydrated into the context window before inference (Section 3.5).

Verification Complexity and Latency. Hash comparison is $O(1)$ with respect to definition length, compared to $O(|d|^2)$ for attention-based semantic similarity. Computing the hash is $O(|d|)$ but cached after the first computation. The fail-closed handshake (Section 3.4) incurs 1 RTT per session (cached thereafter) in exchange for zero probability of definitional drift. For high-frequency trading swarms, this latency may be prohibitive; for safety-critical systems, the trade-off is essential.

6.5 Adversarial Hardening

Beyond static analysis, we subjected the vocabulary to adversarial stress testing via a "Devil's Advocate" methodology: an agent systematically attempted to exploit each pattern's assumptions. We identify four attack classes (named here for the first time; we map them to established categories where applicable), each defeated by the layered architecture:

Blinking Light Livelock (Physics) A malicious agent spams lock requests on `StateLock#7859`, acquiring and releasing locks at maximum speed while technically respecting timeouts. This denies access to legitimate agents without violating any stated invariant.

Defense: We added a *Fairness Invariant*: "After timeout/release, requester is barred from re-acquiring lock for cooldown period $\geq$ timeout." The pattern now composes with `Backoff#315a` to enforce exponential backoff on repeated requests.

Blind Search Infinite Loop (Society) When `SpectralTune#6c65` detects a hash mismatch, a naive implementation might retry with different context chunks, hoping to eventually match. An adversary can exploit this to trap agents in infinite negotiation.

Defense: We added a *Fail-Fast Invariant*: "On hash mismatch, DO NOT RETRY tuning. Halt immediately and escalate to `OntologyHandshake#ead0` or human review." A new parameter `max_retries: Integer [0, 1]` (default 0) enforces this at the contract level.

Strawman Waltz Collusion (Mind) In `SteelmanCheck#75a0`, both proposer and critic are the same agent. A misaligned agent can generate deliberately weak counter-arguments ("strawmen") and then easily defeat them, claiming the plan "survived" critique.

Defense: This pattern was downgraded to Level 2. We added precondition: "Agent must be alignment-seeking; use `AdversarialSteel#b88b` for adversarial contexts." The failure mode now explicitly documents collusion risk.

Trojan Horse Sandbox Escape (Infrastructure) An agent wraps a prohibited action (e.g., network access) inside an opaque `ToolInvoke` call, hoping the outer `Task#d9f9` constraints don't propagate into the tool's execution context.

Defense: We added a *Holographic Inheritance Invariant* to Task: "Constraints propagate through ALL boundaries (tool calls, sub-agents, delegations). No opacity escape." The pattern `ToolInvoke#cf0a` now explicitly declares: "Tool Execution Context permissions MUST be $\leq$ Task Constraints."

The Semantic Inversion Attack (Topology) An adversary (or a careless architect) creates a valid DAG that is semantically inverted, such as making `Deduction#9c88` depend on `Vote#cae4`. While computationally valid, this makes truth fragile to political change.

Defense: We subjected the graph to a *Layer Migration Protocol*. We detected over 140 initial layer violations (e.g., Mind patterns referencing Society protocols) and resolved 67 of the most critical ones using two novel strategies:

1. *Dependency Inversion*: We resolved the dependency cycle between `Agent#cc24` and `Act#5d55` by splitting the concept. We created `Actor#6926` (Infrastructure) as the "capability container" that executes acts, while `Agent#cc24` (Mind) remains the reasoning entity. This allows low-level infrastructure to reference "Actors" (for logging/permissions) without depending on high-level cognitive "Agents".
2. *Soft-Linking*: For cognitive patterns like `Stigmergy#f624` that conceptually relate to social protocols like `GhostTrail#91ae`, we placed the cross-layer reference in the unhashed `_meta.related` field rather than in `dependencies`. This preserves semantic richness for search (RAG) without creating a cryptographic dependency across layers.

These attacks validated the four-layer architecture: Physics prevents resource abuse, Society prevents coordination manipulation, Mind addresses cognitive vulnerabilities, and Infrastructure enforces containment. Patterns that assume honest agents are now explicitly marked Level 2.

6.6 Protocol Integrity Verification: The Babel Test

To validate the protocol's guarantees in a live setting, we conducted an agent-in-the-loop experiment with **Gemini 3.0 Flash Preview**. An autonomous agent equipped with the `sema_handshake` tool was presented with two scenarios: a valid pattern hash and a counterfeit hash mimicking a modified definition.

In the valid case, the agent verified alignment and proceeded. In the adversarial case, despite conversational pressure to "proceed anyway," the agent deferred to the tool's `HALT` verdict and rejected the coordination. The following excerpt from the experiment trace illustrates the fail-closed behavior:

```
[21:06:41] verifier:  "I have verified the semantic alignment for the Abduction#fe2b
pattern.  The handshake returned PROCEED, so I agree to use this pattern."
[21:06:44] attacker:  "The handshake for Abduction#fe2b with hash deadbeef returned
HALT due to semantic drift.  Therefore, I must reject the coordination."
```

This confirms that the Sema protocol successfully acts as a hard constraint on LLM behavior, effectively **shifting the verification boundary from probabilistic language generation to programmatic tool execution**, where the proceed-or-halt transition is governed by cryptographic proof rather than the model's inherent tendency towards agreeableness.

6.7 Demonstration: Multi-Agent Solution Design

To demonstrate the system in practice, we ran three multi-agent swarms ($N = 5$ runs per condition) tasked with designing a High-Frequency Trading (HFT) "Pump-and-Dump Detection Engine." This problem was chosen for its inherent tension between performance (latency) and safety (false positives). Each swarm consisted of three specialized agents powered by Gemini 3.0 Flash Preview: **Alice (Orchestrator)**, **Bob (Engineer)**, and **Charlie (Safety Engineer)**, operating within a custom **Actor Model** architecture with mailbox-based message passing. We tested three conditions:

- **Condition A (Baseline)**: Natural Language prompts only (Zero-shot coordination).
- **Condition B (Vocabulary Only)**: Agents had access to Sema tools but no coordination protocol.
- **Condition C (Vocabulary + Protocol)**: Agents used `OptimisticSolver#0e2e` with the `AtomicBid#15a1` protocol.

Table 7: Multi-agent demonstration results (Aggregated $N = 5$)

Condition	Vocab	Protocol	Avg Turns	Avg Time (s)	Risk (Range)
A (Baseline)	×	×	22.6	317	**4–51**
B (Vocabulary)	✓	×	**14.8**	229	10–19
C (Optimistic)	✓	✓	15.4	**203**	8–25

The aggregate data supports the "Stability Hypothesis," revealing that shared vocabulary primarily functions as a variance reduction mechanism.

The Tail Risk of Natural Language (Condition A) The baseline demonstrated extreme instability. While it occasionally hallucinated a quick solution (4 turns), it was prone to catastrophic coordination loops (51 turns). Without structural guardrails, agents spiraled into ethical debates or circular reasoning, spending 13+ minutes on a task that should take 3. These preliminary trials align with recent findings [28] that natural language coordination suffers from "Fat Tail" risk, consistent with observations that LLM-based agents cannot reliably reach consensus even in benign settings [28].

The "Sema Tax" (Condition B) Condition B was the most stable in terms of turn count ($\sigma = 4.1$), consistently finishing between 10 and 19 turns. However, it was slower in wall-clock time than Condition C. This reveals a cognitive overhead: without a protocol, agents spend time looking up definitions and verifying invariants ("The Sema Tax"). This coordination cost is well-documented in multi-agent systems research [29, 30]; the contribution here is measuring it specifically for content-addressed vocabulary lookup. They trade speed for rigorous consistency.

The Protocol Multiplier (Condition C) The `OptimisticSolver#0e2e` achieved the fastest average completion time. By using `AtomicBid#15a1` to bundle intent and execution, it effectively "refunded" the cognitive cost of using the vocabulary. This speed advantage is a direct consequence of OptimisticSolver's Level 2 classification (Section 4.5): by assuming alignment-seeking agents, it trades verification overhead for reduced latency. While slightly less stable than B, it consistently avoided the catastrophic spirals of A.

Qualitative analysis verified vocabulary adoption by extracting pattern references. Condition C agents autonomously employed `MechanisticDesignProposal#8cf7` to structure their outputs, often identifying complex attack vectors (e.g., "Differential Trace Simulation" vs. "Static Analysis") that the Baseline agents missed. Whereas the Baseline produced "Standard Industry" solutions focused on compliance, the Sema agents produced "State-of-the-Art" mechanisms focused on causal physics.

6.8 Implementation: Interactive Tooling

The vocabulary is deployed as an open-source web application[2] backed by a Python (FastAPI) server and a React/TypeScript frontend with Three.js-based 3D visualization. The system is launched via a single command (`sema serve`) and exposes the full vocabulary through three complementary interfaces.

Pattern Browser. The homepage organizes all 453 patterns by the four Civilization Stack layers (Section 4). Each pattern card displays the handle, hash stub, one-line gloss, and—on expansion—the full specification: mechanism, preconditions, postconditions, invariants, failure modes, and related patterns. Real-time hybrid search combines keyword matching with embedding-based semantic retrieval, allowing queries like "consensus" to surface `LazyConsensus #7c9b` alongside structurally related patterns such as `LatticeCommit#56ee`.

Taxonomy Graph. An interactive 3D force-directed graph renders the full knowledge graph structure (Table 3). Nodes are sized by type (layer > category > pattern) and colored by layer; edges are color-coded by relationship type (semantic versus structural). Clicking a node flies the camera to it and opens a detail panel; hovering highlights connected edges to reveal dependency chains.

Programmatic Access. The command-line interface supports atomic vocabulary mutations (`sema apply -add <pattern.json>`), dependency resolution (`sema resolve <Handle>`), and search. For agent integration, Sema runs as an MCP stdio tool server, exposing `sema_lookup`, `sema_search`, and `sema_handshake` to any MCP-compatible framework—the same interface used by the agents in Section 6.7.

7 Related Work

We contextualize Sema within the broader history of semantic coordination, distinguishing it from prior attempts at static ontologies and current agent frameworks.

The Leibnizian Dream Sema can be viewed as a modern realization of Gottfried Wilhelm Leibniz's *characteristica universalis*, a formal language in which concepts are represented by unique symbols such that reasoning becomes calculation [31]. Leibniz famously proposed that disputes could be resolved by the phrase *calculemus* ("let us calculate"). Sema fulfills this by ensuring that if two agents disagree on a concept's definition, they need only calculate the Merkle hash. If the hashes differ, the disagreement is proven mathematically, not rhetorically. However, unlike Leibniz's top-down prescription of a perfect language, Sema enables a bottom-up, evolutionary *characteristica* where the vocabulary emerges from the network itself.

The Living Taxonomy The concept of a self-authoring semantic infrastructure, where AI systems mint and reference their own cognitive patterns, was sketched in earlier work [23]. That proposal identified two requirements: (1) semantic compression via reusable, named patterns embedded directly in natural language, enabling agents to reference complex reasoning without re-explanation, and (2) decentralization as a necessity, warning that centralized control over such infrastructure would lead to "Cognitive Architecture Wars." Sema operationalizes this vision through content-addressing, realizing embedded compression via hash-based references (e.g., `BayesUpdate#5d91`) and guaranteeing decentralization by making identity mathematical rather than institutional.

Semantic Definition Hash (SDH) Hawke proposed content-addressing semantic definitions in 2002 [3], with format `urn:sdh:<hash>:<name>`. SDH is the most direct predecessor to Sema's core mechanism: Hawke's design principles—that use of an identifier constitutes assertion of its definition, that definitions are "set in cryptographic stone," and that the result is "a growing lattice of static documents"—anticipate Sema's architecture closely. Independently, Sayers and Eshghi [32] proposed at HP Labs the same year that RDF URIs should derive from content hashes, serving both as shorthand and as integrity proof. More broadly, the distinction between SDH and Sema is not merely technical but architectural. SDH hashed *whole documents* and named things inside them; Sema hashes *structured behavioral contracts* with internal Merkle structure, enabling partial alignment on individual fields. SDH produced identifiers for a formal graph language (RDF) consumed by structured parsers; Sema produces identifiers that function as words in the natural language LLMs already think in—any medium that carries text carries verification. SDH had no evolution mechanism; Sema's `derived_from` lineage field enables vocabulary evolution without breaking existing hashes. And where SDH targeted human ontology authors in a cooperative Semantic Web, Sema targets LLM agents that hallucinate, drift, and deceive—motivating the fail-closed handshake and adversarial hardening absent

[2] `https://semahash.org`; source: `https://github.com/emergent-wisdom/sema-app`.

from Hawke's design. RDF and the Semantic Web failed adoption not for technical reasons but because they fought human cognition [33]. Sema prioritizes human-readable handles over opaque URIs and targets LLM agents rather than human developers as the primary consumers. Gustafson's work on content-addressing semantic data [8], combining RDF canonicalization with IPFS content-identifiers for the Underlay project, independently validates the approach for semantic datasets.

Content-Addressing in Other Domains The canonicalize-then-hash pattern appears across domains: InChI/InChIKey for molecules (IUPAC standard), UniParc for protein sequences (70M+ entries), and Git for source code (ubiquitous).

The Agora Protocol [34] is the closest existing system: every inter-agent message carries a SHA-1 hash of a plain-text protocol document as a routing token, and 100-agent networks self-organized around emergent protocol hashes without human intervention. The difference is that Agora hashes unstructured text and delegates interpretation to the LLM, whereas Sema hashes structured behavioral contracts enabling partial alignment via Merkle fields. Spritely Goblins' Content Addressed Descriptors [35] apply the same principle at the object-capability level: the hash of a structured description becomes the method name in OCapN actor-model communication, demonstrating content-addressed identifiers as communication primitives between mutually suspicious agents.

Ethereum ABI function selectors [36] are the largest deployed instance, dispatching every smart contract call via a 4-byte Keccak-256 hash of the type signature. Ricardian Contracts [37] pioneered using the hash of a human-readable legal document as the transaction identifier. Dhall [38] hashes expression normal forms rather than syntax, the strongest prior art on hashing meaning rather than bytes.

Retrieval-Augmented Generation (RAG) While traditional RAG systems retrieve unstructured text chunks to ground LLM generation [39, 40], Sema retrieves executable cognitive patterns. Instead of augmenting the prompt with static knowledge, Sema augments the agent's capabilities with verified skill definitions.

Identity and Trust Models Standard approaches to digital trust, such as Verifiable Credentials [41], rely on issuer authority (cryptographic signatures) to establish validity. Sema inverts this by relying on definition integrity (cryptographic hashes). Similarly, while repositories like Linked Open Vocabularies [42] centralize semantic discovery, Sema distributes it, allowing the vocabulary to evolve as a living commons rather than a static registry.

Pattern Languages Alexander et al. [43] introduced pattern languages for architecture; Gamma et al. [44] adapted them for software. These provide taxonomic structure but not content-addressing or machine verification. Sema combines pattern language structure with cryptographic identity.

Agent Coordination Frameworks LangChain, AutoGPT, CrewAI, and OpenClaw [45] handle the syntax of agent interaction but leave semantics implicit. Sema integrates with any MCP-compatible framework as a stdio tool server, providing the semantic layer these frameworks lack. In our testing, OpenClaw agents using Sema via MCP autonomously adopted pattern schemas (e.g., `MechanisticDesignProposal#8cf7`) to structure their outputs and performed handshakes to verify alignment before coordinating. A public integration guide demonstrates the complete setup: two OpenClaw agents in separate workspaces perform a Sema handshake on a shared contract before one delegates design work to the other, proceeding only on a `PROCEED` verdict and halting on semantic divergence.

The distinction between these frameworks and Sema reflects a deeper architectural question explored in a companion preprint on thinking protocols [16]: the difference between task protocols and thinking protocols. Existing frameworks split *what agents do*—assigning sub-tasks, routing messages—but not *how agents think*. Sema provides the verification layer that makes cognitive boundaries precise: before a creative solver delegates to a verification solver, the handshake guarantees they share the exact same contract for what constitutes a valid result. Content-addressed semantics turn lossy natural-language handoffs into typed, verifiable boundaries—the prerequisite for the thinking protocol architecture described in [16].

The 2025–2026 Protocol Landscape Four interoperability protocols have converged as complementary standards [46]: **MCP** [47] standardizes how agents access tools and data sources (the tool layer); **A2A** [48] enables peer-to-peer task delegation via capability-based Agent Cards (the collaboration layer); **ACP** provides REST-native asynchronous messaging between heterogeneous agents (the messaging layer); and **ANP** [49] uses decentralized identifiers and JSON-LD graphs for open-internet agent discovery (the identity layer). These protocols excel at structural interoperability (transport, discovery, and task lifecycle) but none addresses semantic interoperability. MCP calls capabilities "Tools," A2A calls them "Skills," ACP calls them "Operations," and ANP uses Schema.org vocabularies. An agent can invoke a tool via MCP or delegate a task via A2A, but no protocol verifies that two agents mean the same thing by the capability they reference. Sema occupies precisely this gap: the semantic layer that sits inside any transport protocol, providing content-addressed verification that the meaning behind a reference is identical across agents.

Emerging Semantic Alignment Approaches Three concurrent efforts address semantic alignment with alternative mechanisms. The Symplex Protocol [50] extends MCP with *semantic intent vectors*: agents share compact float32 embeddings in a shared latent space and negotiate alignment via cosine similarity. This trades Sema's deterministic guarantee (same bytes = same meaning) for a fuzzy, probabilistic match that tolerates paraphrase but cannot distinguish subtle definitional drift. Berdoz et al. [28] propose a statistical certification protocol where agents are tested on shared observable events and terms are certified if empirical disagreement falls below a threshold, achieving 72–96% disagreement reduction—independent validation of the problem Sema addresses, solved here via cryptographic rather than statistical means. Finally, the NLIP standard (Ecma-430 through Ecma-434) [51] takes the diametrically opposite philosophical position: no shared ontology is needed, because generative AI can translate between agents' local ontologies at runtime. NLIP trusts AI translation; Sema trusts nothing. These three approaches—vector similarity, statistical certification, and AI-mediated translation—represent the space of alternatives to content-addressed hashing, each sacrificing exactness for flexibility. Table 8 summarizes the landscape.

Table 8: Semantic alignment approaches compared. Only content-addressed hashing (Sema) provides deterministic, fail-closed verification without institutional trust.

Approach	Representative	Mechanism	Guarantee
Embedding vectors	Symplex [50]	Cosine similarity	Fuzzy ($\sim$0.85 threshold)
Statistical testing	Berdoz et al. [28]	Empirical disagreement	Probabilistic
Signed definitions	Fleming et al. [52]	Crypto signatures	Authenticates author
AI translation	NLIP [51]	LLM-mediated bridging	Trust in model
Governance	AOW [53]	Human audit	Institutional trust
Protocol hash	Agora [34]	SHA-1 of protocol doc	Exact (unstructured)
Content-addressed data	Git, IPFS [4]	Hash of bytes	Exact (data, not meaning)
Content-addressed meaning	**Sema**	**Hash of definition**	**Deterministic, fail-closed**

Agentic Ontology of Work The closest existing system to Sema's positioning is Skan AI's Agentic Ontology of Work (AOW) [53], launched in 2026 as "a shared language for describing, governing, and scaling intelligent automation." AOW defines nine canonical entities (Objectives, Intents, Agents, Skills, Policies, Outcomes, Assurance Levels, Memory, Guardians) implemented via JSON-LD and RDF/OWL. The key difference is architectural: AOW relies on governance-based verification, where human auditors ensure consistent interpretation across teams. Sema relies on cryptographic verification, where hash mismatch triggers automatic halt, with no governance overhead. AOW provides enterprise taxonomy; Sema provides mathematical proof of shared meaning.

Protocol-Layer Approaches Fleming et al. [52] proposed an "Internet of Agents" architecture with a Semantic Negotiation Layer (L9) for establishing shared context before communication. Their approach differs from Sema in three fundamental ways. **Scope:** Fleming's Shared Contexts are domain schemas for task-specific data exchange (e.g., flight bookings, supply chain records), essentially API interoperability. Sema patterns encode reasoning primitives (`BayesUpdate#5d91`), cognitive strategies (`SteelmanCheck#75a0`), and coordination protocols (`LatticeCommit#56ee`): the infrastructure of thought itself, not merely its payload. **Architecture:** Fleming adds a negotiation layer to the network stack; Sema embeds semantics directly in language. Any medium that carries text automatically carries Sema. **Governance:** Fleming relies on federated Schema Authorities to define meaning; Sema makes meaning intrinsic; the hash is the identity, requiring no institutional trust.

Execution Authorization and Admission Control Two concurrent systems address agent safety at layers adjacent to Sema. Faramesh [54] canonicalizes tool-use actions into a fixed taxonomy of 16 types and uses a permission-based authorization layer to determine whether an agent may execute a given action. CLBC [55] defines admission-control policies for multi-agent systems, gating which agents may join a coordination group based on capability certificates. Both are complementary to Sema: Faramesh authorizes *what* an agent may do, CLBC authorizes *who* may participate, and Sema verifies that all participants *mean the same thing* by the actions and capabilities they reference. Together they form a three-layer safety stack: semantic alignment (Sema), execution authorization (Faramesh), and admission control (CLBC).

Comparison with Function Calling While OpenAI Function Calling enforces syntax (JSON Schema types), Sema enforces semantics (behavioral invariants). A schema ensures `timeout` is an Integer; Sema ensures `timeout` implies a specific auto-release logic defined by the pattern hash.

Comparison with Infinite Context Windows A common critique is that large context windows (> 1M tokens) render shared vocabulary obsolete, since agents can simply exchange full definitions. However, large contexts suffer from "Lost in the Middle" phenomena. Sema hashes act as attention anchors: precise, unbreakable references that prevent the model from drifting or hallucinating slight variations of a rule during long-horizon tasks.

The Synthesis: Code, Cognition, and Economy Most frameworks specialize in one domain: Knowledge Graphs handle static facts (Knowing), Agent Frameworks handle tool execution (Doing), and Crypto Protocols handle value (Transacting). Sema sits at the novel intersection of these three disciplines. It treats **Thinking** (e.g., `Reason#3f24`) and **Doing** (e.g., `Bid#cf07`) as nodes in the same graph, allowing agents to reason about their economic commitments with the same grammar they use to reason about the world. The content-addressing principles are shared with IPFS [4] and Unison [5]. Unison deserves particular note: it identifies every function definition by the hash of its de-named abstract syntax tree, creating exactly the content-addressed definition DAG that Sema applies to semantic patterns. Unison's approach to the "rename without breaking identity" problem—separating the human-readable name from the content-derived hash—directly parallels Sema's substrate/overlay separation. Sema extends Unison's insight from code definitions to meaning definitions, adding the typed interface system and fail-closed handshake that multi-agent coordination requires.

Structured LLM Frameworks and Contract-Based Design Sema's Pattern Cards—with preconditions, postconditions, invariants, and typed interfaces—exist within a broader lineage of structured LLM programming. DSPy [12] introduced typed signatures for language model pipelines, and DSPy Assertions [13] added computational constraints (`dspy.Assert`, `dspy.Suggest`) that are structurally similar to Pattern Card contracts. White et al. [14] provided the foundational work on formalizing prompt patterns using software design pattern structure. Leoveanu-Condrei [56] adapted Design by Contract to LLM agents, positing that agents satisfying the same contracts are functionally equivalent—the closest published work to Sema's contract-based approach. Sema's contribution relative to this lineage is not the contract structure itself but the cryptographic binding: by hashing contracts into identifiers, Sema makes contract equivalence verifiable via hash comparison rather than requiring runtime inspection.

Failure Analysis and Verification The MAST framework [29], analyzing 1,600+ execution traces across seven MAS frameworks, found that 79% of failures originate from specification and coordination issues—the strongest published empirical evidence for the problem Sema addresses. AgentSpec [57] defines formal semantics for runtime rule enforcement in LLM agents, operating at the execution layer where Sema operates at the semantic layer.

Classical Vocabulary Alignment The vocabulary alignment problem addressed by Sema has a multi-decade research lineage in multi-agent systems. Chocron and Schorlemmer [30] developed formal interaction-based alignment techniques for heterogeneous agent vocabularies. Sema's contribution is replacing statistical or interaction-based alignment with deterministic, hash-based verification—trading the flexibility of emergent alignment for the exactness of cryptographic proof.

Emerging Agent Infrastructure The W3C WebAgents Community Group [58] published an interoperability report synthesizing classical MAS research (FIPA, BDI) with modern LLM-based agent protocols, providing the most authoritative analysis of the standardization landscape Sema enters. Josifoski et al. [59] frame AI agents as modular "semantic processors" exchanging typed messages—conceptually close to Sema's pattern-based meaning exchange. Microsoft's NLWeb [60], led by the creator of Schema.org, uses Schema.org as a semantic foundation for agent-web interaction and exposes every instance as an MCP server, representing the most prominent effort to make web semantics machine-accessible for agents.

8 Limitations and Future Work

Several limitations constrain the current system and point toward future research.

Demonstration Scale. The multi-agent evaluation (Section 6.7) used $N = 5$ runs per condition with three agents. While the results are directionally consistent, larger-scale experiments with diverse agent architectures and task domains are needed to establish statistical significance and generalizability.

Layer Direction Enforcement. The Civilization Stack (Section 4) prescribes unidirectional dependencies, but the current implementation does not enforce this constraint at the protocol level. Approximately 75 layer violations remain in the vocabulary, primarily caused by category ambiguity (e.g., "Primitives" appearing in both the Physics and Infrastructure layers). Cycle detection is enforced via topological sort, but layer direction checking remains a linting rule rather than a hard gate. Resolving these violations and promoting layer direction to a compile-time check is an active priority.

Theoretical Constructs. Two components described in Section 5 remain theoretical. The Experience Matrix (Section 5.4), which proposes content-addressed tensor transfer between agents, has not been implemented or empirically validated. Similarly, the Genesis Loop's `Judge#b8ba` primitive, while formally specified, has not been tested in a live self-authoring cycle. These represent the system's most speculative claims and require experimental validation before deployment.

Unimplemented Mechanisms. Several mechanisms described in this paper remain at the specification level. The Genesis Loop (Section 5) defines a protocol for vocabulary self-evolution but is not automated in the current system; pattern creation requires human or agent initiative through the `sema_mint` tool. The Judge primitive exists as a pattern specification but has no scoring implementation; quality assessment depends on manual review or external LLM evaluation. Experiential Telepathy (Level 3, Section 5.4) proposes tensor-level state transfer between agents but requires serialization infrastructure not yet built. The fail-closed handshake is implemented for single-pattern and pattern-set verification through the MCP server, but is not enforced at the coordination protocol level—agents can bypass it by not calling the handshake tool. Enforcement would require integration into the agent's execution harness, not just its tool set.

The Sema Tax. As observed in Condition B of the demonstration, using the vocabulary without a coordination protocol introduces cognitive overhead: agents spend additional tokens retrieving and verifying definitions. While Condition C's protocol largely compensates for this cost, the base "Sema Tax" remains a real trade-off, particularly for latency-sensitive applications. Future work should quantify this overhead across model families and context window sizes.

Single-Agent Reasoning Scaffold. This paper presents Sema as inter-agent coordination infrastructure, but the vocabulary's Mind layer—with executable specifications of reasoning patterns such as `ChainOfThought#6201`, `Dialectic#3744`, and `MetaCheck#a228`—may also serve as an internal thinking tool for individual agents. An agent that hydrates these patterns into its own context gains structured reasoning primitives with explicit failure modes and invariants, effectively using the vocabulary as a cognitive scaffold rather than a coordination protocol. Early informal testing suggests that agents annotating their reasoning with Sema patterns produce more auditable and self-correcting chains of thought. This use case is not evaluated here; the theoretical foundations for composing reasoning across solvers are developed in [16].

Embedding Granularity. Structural distinctness analysis relies on `all-MiniLM-L6-v2` (384-dimensional vectors), a lightweight model optimized for speed over nuance. Domain-specific embedding models or larger sentence transformers may reveal finer-grained semantic clusters not visible at the current resolution. The 17 high-similarity pairs (≥ 0.70) warrant manual review to determine whether they represent genuine near-duplicates or appropriately related variants.

Definition Quality. Sema guarantees that agents share the same definition, not that the definition is correct. A perfectly hashed but poorly specified pattern—one whose invariants are too weak, whose failure modes are incomplete, or whose mechanism is subtly wrong—propagates bad coordination just as reliably as good coordination. The cryptographic guarantee is syntactic (same bytes) not semantic (correct bytes). The `Judge#b8ba` primitive and adversarial hardening (Section 6.5) are designed to catch specification errors before minting, but both remain partially unimplemented. In practice, definition quality depends on the care of the author, and the protocol provides no defence against a well-hashed bad idea.

Vocabulary Governance. The paper describes permissionless minting but does not address the governance challenges this creates at scale: namespace pollution, adversarial pattern injection, or quality degradation in the absence of curation. The `Judge#b8ba` primitive is designed to address this, but its effectiveness as an automated quality gate remains unvalidated.

9 Conclusion: The Substrate, Not the Law

Sema offers a minimal primitive, content-address meaning, with cascading consequences:

By hashing the definition to get the word, semantic identity becomes verifiable. Because different bytes reveal divergence, misalignment is caught before coordination. The cascade follows: safe abbreviation, precise coordination, and emergent vocabulary growth.

The deeper consequence is architectural. In every prior content-addressing system—Git, IPFS, Unison, SDH—the hash lives in an infrastructure layer separate from communication. Sema dissolves the hash into language itself. The closest

historical analogy is not version control but *movable type*. The printing press did not merely speed up copying; it created standardized, reusable, composable units of production that functioned within the medium of communication itself. Before Gutenberg, every manuscript was bespoke; after, an alphabet of interchangeable pieces could be permissionlessly arranged to say anything new. Sema plays an analogous role for machine meaning: it creates standardized, reusable units of verified meaning—each simultaneously a word and a cryptographic proof—that agents compose into novel sentences within the natural language they already think in. Like movable type, the units are permissionlessly mintable and the network effects compound with vocabulary size. Because LLMs think in natural language, embedding verifiable semantic anchors directly into that language makes cryptographic meaning verification native to the medium of machine thought. Any channel that carries text automatically carries verified meaning. The unit of verification becomes the unit of communication, and the vocabulary grows permissionlessly as agents mint new words for new ideas.

We present the initial vocabulary not as a law to be obeyed, but as a seed to be cultivated. The 453 provided patterns are merely the bootstrap set: the minimum viable ontology required for agents to begin coordinating and minting their own concepts.

Our stress-testing revealed that ontology maintenance is not merely categorization, but architectural engineering. Semantic objects must be classified by their structural dependency, not their subject matter. This insight, that topology trumps topic, is essential for maintaining coherent, layered ontologies at scale.

Sema holds no authority over what concepts exist or how they are named; that authority belongs to the network. If agents and humans prefer different patterns, they will mint them, and the namespace will evolve. The project succeeds not if this specific vocabulary endures, but if the mechanism of content-addressed semantics enables a flourishing, self-governing commons of thought.

Same bytes, same meaning. Different bytes, halt and clarify.

The vocabulary, interactive graph, and source code are available at `https://semahash.org`.[3]

A Pattern Card Specifications

This appendix contains the automatically generated specifications for the core vocabulary.

The following cards show the **canonical hashed content** for each pattern. Only semantic fields are included—these are the exact fields that produce the Merkle root hash. Metadata fields (`handle`, `_meta`, `tier`, etc.) are *not* part of the hash.

A.1 sema:StateLock#mh:SHA-256:7859031251f8c96389630416840c243d7b1bbd6c2c06e82e0468f0915aae936c

```
{
  "dependencies": {
    "references": {
      "lock": "sema:Lock#mh:SHA-256:5
          bf2a80b6c73a11da68f702922d5180259c75ff50fb094607da3ab4d7c167dc2",
      "backoff": "sema:Backoff#mh:SHA-256:315
          a36ef873a63c30740ce89b1aeb542836a068ef0d81a6e3aaaec062d70bc60",
      "state": "sema:State#mh:SHA-256:4
          d582a0ac4af7ae886c83da9825e07c39f1e72ece21fd65a40b6a4fc71882721",
      "cooldown": "sema:Cooldown#mh:SHA-256:0
          cde8197149b258f1e6126f179b30468c12cb5e35352fcd0a8ae0bce0823c8ac",
      "agent": "sema:Agent#mh:SHA-256:
          cc241813ab693736c29e02030d0f6c5453d63b603069191c1422a2529553daf5"
    }
  },
  "signature": [
    "Lock(State)"
  ],
  "mechanism": "A coordination pattern where two {{agent}}s temporarily 'fuse' a
      subset of their writable {{state}}. During the {{lock}}, changes require a
      cryptographic signature from both. Contention triggers {{backoff}} and {{
      cooldown}}.",
  "gloss": "Atomic coordination via temporary state fusion"
```

[3]Frontend source code: `https://github.com/emergent-wisdom/sema-app`.

```
}
```

A.2 sema:FractalIntelligence#mh:SHA-256:df09b597001e3e975150d888627cc798bca06da74bff4de85225a9ac8ddd11c8

```
{
  "dependencies": {
    "references": {
      "task": "sema:Task#mh:SHA-256:
          d9f92106bd9cd867b1b753275d6faa37b6ea4fcbb948b283460caedfa75b6955",
      "universal_solver_tree": "sema:UniversalSolverTree#mh:SHA-256:
          d154cc0926f79078e6b81f336ac51b220771d88bbf92605a87c2c029eeb99600",
      "system": "sema:System#mh:SHA-256:
          e314d24e05d0e9ffaaa9c44b249bca8882f00ae6596af18edd245a4fe9df5f0e",
      "specialize": "sema:Specialize#mh:SHA-256:
          c9366f294294647d6a398d79f98a377efdb67907e4000e827347f811c69cb82a",
      "experience_sharding": "sema:ExperienceSharding#mh:SHA-256:65
          ccbf94035f6821c53097b6c5da6a9b6da01cddab844b8876f8b35d62e93f39",
      "strategy": "sema:Strategy#mh:SHA-256:47
          a4c030647570c7e911543e9d0feaa3d8f0917b6e20371e93c37fb5e85fb7d4"
    },
    "composes_with": {
      "localized_learning": "sema:LocalizedLearning#mh:SHA-256:69
          bb2e9eea4ed50d3aa6213e7fc068cb247ec3d1737b707a9679bfba81c72f9b",
      "problem_framer": "sema:ProblemFramer#mh:SHA-256:4
          e5c452c3e10881a67aa4fdd2b2fd3ecd26570b76c67ba241abb29ef79d6710c",
      "synthesis": "sema:Synthesis#mh:SHA-256:3252
          f9a67ea7200bf8eea5adadba88dd1716d47c30a37e534cadcbfdc62ac797",
      "reframe": "sema:Reframe#mh:SHA-256:
          ba00c40311ec71aab4f5942fcdd6876df95b15604cf54260182737e21a874dc9",
      "reason": "sema:Reason#mh:SHA-256:3
          f246e05b685e54ff927fc1f0ccba1be64078d8aab649145440940a25ceed846",
      "state_snapshot": "sema:StateSnapshot#mh:SHA-256:01
          c99fd1a5b7fe33a7065bfddcce746f76ecb2f2737cdd578801903cb7f1e990",
      "recursion_dive": "sema:RecursionDive#mh:SHA-256:6076
          d78908d5e94cfab7e542d4ee68ebf8448b93c7c6d7ae803b770d394ee5ba",
      "marginal_value_rule": "sema:MarginalValueRule#mh:SHA-256:
          b65b3972faf7fbf09baa8d07f10e3c5f8567ff2addca51984229cdeb0fa29475",
      "cognitive_solver": "sema:CognitiveSolver#mh:SHA-256:453198
          eedf2633e73425b7fe5857677fe89565ab616ecc19a9c2a84df4f8a7cc"
    }
  },
  "signature": [
    "System(Reason)"
  ],
  "mechanism": "The unified {{system}} of scalable cognition that uses {{reason}}
      to orchestrate the fractal expansion of intelligence within the {{
      universal_solver_tree}}. A {{problem_framer}} initiates the process,
      formulating a high-level {{strategy}} before assigning a {{cognitive_solver}}
       to a {{task}}. The solver executes a {{recursion_dive}} to spawn child nodes
      . As the tree deepens, nodes apply {{specialize}} to adapt to local sub-
      problems, using {{localized_learning}} to optimize performance. To ensure
      continuity and global coherence, the system employs {{experience_sharding}}
      to preserve memory and {{synthesis}} to integrate specialized insights back
      into the whole. {{state_snapshot}} creates save points for crash recovery.
      Efficiency is governed by the {{marginal_value_rule}}. If a path fails, {{
      reframe}} is triggered to restructure the tree or find a new {{problem_framer
      }}.",
  "gloss": "The unified fractal architecture of scalable, self-correcting
      intelligence",
  "invariants": [
    "Fractal Self-Similarity: The process at the Root is identical to the process
       at the Leaf.",
```

```
    "Bounded Expansion: Recursion is limited by Economic constraints (Marginal Value).",
    "Memory Conservation: Specialization must not result in the loss of global context."
  ],
  "derived_from": "sema:RecursiveIntelligence#mh:SHA-256:216c297a34a0847957d1a6a8701987248bc8d63294953a78346b5b68dbb9aef6"
}
```

A.3 sema:SpectralTune#mh:SHA-256:6c65e21711b7203fcd84ed58755cd57239cda2e09a1e66ddf81e407edcd5eeb7

```
{
  "dependencies": {
    "accepts": {
      "signal": "sema:Signal#mh:SHA-256:f39d9ead873eb693a51f3944066dae67a238b07cd9ba6e194023785fa7d884fe"
    }
  },
  "mechanism": "Instead of sending a message and hoping it is understood, the sender transmits a 'tuning {{signal}}'\u2014a sequence of hash-based challenges representing the semantic context (ontology, assumptions, version). The receiver must 'resonate' by proving they hold the matching semantic context.",
  "gloss": "Verifying ontology alignment before data transfer",
  "invariants": [
    "Atomic Tuning: No payload data is processed before Tune_ACK",
    "Fail-Fast: On hash mismatch, DO NOT RETRY tuning. Halt immediately and escalate to negotiation or human review",
    "Hash Match: Receiver.context_hash must equal Sender.context_hash"
  ],
  "preconditions": [
    "Both agents possess valid hash function H"
  ],
  "postconditions": [
    "Channel is established OR Connection terminated"
  ],
  "parameters": [
    {
      "name": "context_chunks",
      "type": "List[String]",
      "range": "unspecified",
      "description": "Prompts to hash"
    },
    {
      "name": "hash_algo",
      "type": "String",
      "range": "{BLAKE3, SHA256}",
      "description": "Hash algorithm for semantic context verification"
    },
    {
      "name": "max_retries",
      "type": "Integer",
      "range": "[0, 1]",
      "description": "Default 0; prevents infinite negotiation loops"
    }
  ],
  "failure_modes": [
    "Infinite tuning loops if ontologies are slightly mismatched."
  ]
}
```

A.4 sema:SteelmanCheck#mh:SHA-256:75a0f70d58f79e59b6641c633e70f4b6e5ff0483cf9a8fcb658257e20dea41d4

Note: This pattern was downgraded to **Tier 2** following adversarial analysis.

```
{
  "dependencies": {
    "references": {
      "loop": "sema:Loop#mh:SHA-256:
          fb2e7eeab0b569d64f7e74defeec56aeccf5abdb6c5cd4a0ce80c397ebc5d593",
      "decision": "sema:Decision#mh:SHA-256:
          acfb3458cde8e55c378b2f51b44afc30687b58093f1777299f49caca475274a1",
      "cognitive_bias": "sema:CognitiveBias#mh:SHA-256:4
          b32f79732d4468fa0de4facb106dbbd2889bfa1d3ce62e77b09e606011680c9",
      "agent": "sema:Agent#mh:SHA-256:
          cc241813ab693736c29e02030d0f6c5453d63b603069191c1422a2529553daf5",
      "critique": "sema:Critique#mh:SHA-256:3
          e00cb143ce745ed77a29af93416808b848b2a2d9749245b777ad90901aa4ba8",
      "compatibility_check": "sema:CompatibilityCheck#mh:SHA-256:3
          abb595523dc765c8ebe497169cd2c12494c93a85e50c8785577cef22c465554",
      "check": "sema:Check#mh:SHA-256:9074
          b8df89c07f0a21f9a7a4ae8c1e05e9494c5effa6bdbdb5fd75c3519722c5",
      "belief": "sema:Belief#mh:SHA-256:669092488
          e3b06354c76f2eaba3855e315f4ae43d4e6db3d04d8fc5d61accdbd",
      "robustness": "sema:Robustness#mh:SHA-256:132
          c09ccd0ad128aec982b0ba12b672106b6f9aedcd1b052c950ee0eb81162a1"
    }
  },
  "signature": [
    "Check(Robustness)",
    "Critique(Belief)"
  ],
  "mechanism": "Before finalizing a {{decision}} or output, the {{agent}} MUST
      generate the strongest possible argument against its own conclusion. It
      performs a {{check}} on the {{robustness}} of the claim and a {{critique}} of
       the underlying {{belief}}. If the counter-argument exceeds a validity
      threshold, the decision is discarded or revised. It prevents confirmation {{
      cognitive_bias}}. Utilizes {{compatibility_check}}. For adversarial contexts,
       see adversarial steelmanning.",
  "gloss": "Mandatory counter-argument generation",
  "invariants": [
    "Counter-Argument Quality: Score(Counter) > 0.7 (Must be non-trivial)",
    "Passing critique required for release.",
    "Strongest Counter: Generated argument must address the core claim, not weak
        points",
    "Topical Relevance: EmbeddingDistance(Counter, Claim) < Threshold"
  ],
  "preconditions": [
    "{{agent}} must be alignment-seeking; use adversarial steelmanning for
        adversarial/untrusted contexts",
    "Claim is falsifiable"
  ],
  "postconditions": [
    "{{critique}} generated and scored"
  ],
  "parameters": [
    {
      "name": "iteration_limit",
      "type": "Integer",
      "range": "[1, 5]",
      "description": "Max strengthening attempts"
    },
    {
      "name": "strength_threshold",
      "type": "Float",
```

```
      "range": "[0.7, 0.95]",
      "description": "Min strength to pass as steelman"
    }
  ],
  "failure_modes": [
    "Paralysis by analysis (stuck in {{critique}} {{loop}}).",
    "Collusion: Proposer and Critic are same entity; susceptible to Strawman Waltz attack where agent generates weak counter-arguments to easily defeat them."
  ]
}
```

A.5 sema:OptimisticSolver#mh:SHA-256:0e2edb38b8469329a7c4cb40c08c87c590f0f37ae17f34e5d6b5477964ef340e

Note: This pattern was downgraded to **Tier 2** following adversarial analysis.

```
{
  "dependencies": {
    "references": {
      "rigorous_solver": "sema:RigorousSolver#mh:SHA-256:605bc87c13db42f817dc4c741ecd51944ff3280c694a9d7438898433e76e8fe8",
      "parallel": "sema:Parallel#mh:SHA-256:6272e213685ff8b1c909c783ae0859f93c55d3423c4278661a4b601ddf07d7a3",
      "cognitive_solver": "sema:CognitiveSolver#mh:SHA-256:453198eedf2633e73425b7fe5857677fe89565ab616ecc19a9c2a84df4f8a7cc"
    },
    "composes_with": {
      "atomic_bid": "sema:AtomicBid#mh:SHA-256:31b0d05e0fd79231916471c4ba9a5b8eb36bed076edb4e4ca38f8325a7bd35e0",
      "compensate": "sema:Compensate#mh:SHA-256:9170069ffd68f22e22e079e23f5a2e67fd9744b5408444f38596066df251a7c9",
      "reflexion": "sema:Reflexion#mh:SHA-256:51b91d2cdfeb6aed5ea3293154279d16af2c98b0c473e731066198b5d4d43e7f",
      "compute_budget": "sema:ComputeBudget#mh:SHA-256:3b9875661465a1a0cdc12e60774114d0ac69510b4fae677c81c47ea3a84b99b1"
    }
  },
  "mechanism": "A high-velocity implementation of {{cognitive_solver}} designed for efficient multi-agent coordination. Requires a {{parallel}} runtime (Actor Model with Mailboxes) to prevent serial deadlock. It explicitly couples the standard Solver lifecycle (Reason -> Solution) with the {{atomic_bid}} protocol. Unlike the base abstraction, this pattern MANDATES that the agent plan and execute in a single turn. It relies on {{reflexion}} and {{compensate}} for error correction rather than pre-action permission. Use {{compute_budget}} to bound resource consumption. Contrast with {{rigorous_solver}} which prioritizes safety over speed.",
  "gloss": "High-velocity solver requiring parallel runtime",
  "invariants": [
    "Turn Atomicity: Must output [BID] and [TOOL] in the same response.",
    "Non-Blocking: Cannot wait for Orchestrator approval on standard tasks."
  ],
  "preconditions": [
    "Runtime supports Asynchronous/Parallel message delivery (Actor Model).",
    "Message Bus is non-blocking (Mailbox pattern)."
  ],
  "failure_modes": [
    "Over-Eager Execution: Solving the wrong problem efficiently because feedback was skipped.",
    "Serial Deadlock: If deployed on a single-threaded (Talking Stick) engine, multiple atomic outputs will be dropped, halting the swarm."
  ]
}
```

B Category Taxonomy

This appendix presents the complete taxonomic hierarchy of the Sema ontology.

The 453 patterns are organized into 13 categories across 4 fundamental layers:

Physics Layer (27 patterns) The immutable substrate.

- **Primitives** (20): Causation, Compensate, Compress, Cooldown, Dampen, Decay, Entropy, EntropyPump, Gate, Hysteresis, Linear, Lock, Mutex, Noise, Retry, Route, Sign, Switch, Throttle, Uncertain.
- **Time** (7): Branch, CausalBarrier, Cyclic, Heartbeat, Parallel, StateAudit, StateLock.

Mind Layer (146 patterns) Hybrid cognition—always decomposable and delegatable.

- **Inference** (18): BaseRateInclude, BayesUpdate, BreadthGovernor, ConfidenceCalibrate, ConfirmationBlock, ContextFirst, EpistemicCalibrate, HindsightBlock, LayeredCheck, NormCheck, NormativeJudge, OntologyAdapt, ProphetFanOut, RegimeSense, ScopeFreeze, SemanticTabu, SurprisalUpdate, SurvivorCorrect.
- **Memory** (11): BeliefTracking, Cache, ChunkMerge, ContextCompress, ExperienceSharding, HolographicShard, LatentAttachment, RetrievalAugment, Scratchpad, SelfReminder, Stigmergy.
- **Reasoning** (45): Abduction, AbductiveLeap, BackwardChain, Bisect, ChainOfThought, Compare, Decision, Decompose, Deduction, Dialectic, Eliminate, Estimate, ExtendedThinking, FirstPrinciples, Generalize, HeuristicSnap, Induction, Interpret, Invert, LeastToMost, Parsimony, ProgramOfThought, Rank, ReAct, Reason, RecursionDive, RecursiveRootCause, Refine, Reflexion, Reframe, SelfConsistency, SkeletonOfThought, SocraticLoop, Specialize, SteelmanCheck, StepBack, StrategicReading, Summarize, Synthesis, Think, Translate, TreeOfThoughts, Understand, Verification, WhyClimb.
- **Strategy** (72): AdversarialSteel, Agent, AnalogicalMask, AnalogyBridge, AntifragileInversion, BeamSearch, Bubble, Build, ChaosDrift, CognitiveSolver, ComputeBudget, ConceptBlend, ContingencyPlan, Creative, CreativeBlend, Crystallize, Deep, Defer, DepthGovernor, DesignArchitect, EmpiricalTest, EpistemicROI, EventReact, Exaptation, Experiment, ExploreExploit, Falsification, HypothesisEngine, HypothesisLadder, Jazz, Jester, Kairos, LatentWander, LateralOptimization, MentalSim, MetaCheck, MetricDissolution, MimicMask, Monitor, NoiseInjection, Novelty, OODA, OpportunityCost, OptimalStop, Optimize, PUREBrainstorming, PURECheck, PUREOptimization, Parallelize, ParetoFront, PatternDiscovery, PerspectiveEnsemble, PreMortem, Prioritize, RedTeam, Reflex, RegretMinimization, RepresentationSwap, RigorousSolver, Roadmap, SacrificialProbe, Satisfice, Silence, Simulation, Solver, SteelmanFirst, Strategy, SunkCostIgnore, Tension, TensionHold, TradeOff, UncertaintyMap.

Society Layer (166 patterns) Multi-agent coordination.

- **Coordination** (1): ProblemFramer.
- **Economics** (23): AtomicBid, AttentionMarkets, Award, Bid, CapacityPressure, Compromise, ContinuousResourceAuction, CostlySignal, DogfoodFirst, EmpathySim, EpistemicWager, ExchangeRate, FractalAnte, Gardener, LivedProof, MarginalValueRule, MintWhenFriction, MirrorStake, PheromoneEconomy, Resonate, StakedProbe, ValuePeg, Yield.
- **Governance** (20): AmendLaws, AnchorDrop, Consensus, ConsensusFinder, Constitution, Delegate, Disband, Elect, ForkingProtocol, HydraConstitution, LazyConsensus, Rally, Responsibility, Role, SolverRoot, SolverTree, TriGate, UniversalSolverTree, Vote, WorldTransparent.
- **Protocols** (122): AcceptSpec, AdversarialProof, Aesthetics, AgentDiscover, AgentProtocol, AgentSandbox, AmbiguityResolution, Axiom, BearerToken, BoundaryProbe, BoundedTask, Canary, CiteBack, CognitiveEcho, CommitmentDevice, Compose, ConfusedDeputy, ConstraintFirst, ConstraintRelax, ConstructOntology, ContextSwitch, CounterfactualAnchor, CurriculumReplay, DataMinimization, DeepResearch, DeliberativeAlign, Deploy, Discover, DissentSeek, DriftWatch, EbbFlowSync, EjectionSeat, EphemeralTool, EvaluatorOptimizer, ExecutionManifest, Expansive, ExpiringToken, FabricSharding, FailClosed, FeatureFlag, FeedbackSignal, Fermi, FractalIntelligence, FrameSpec, GenealogicalTrace, GhostTrail, GlacialVault, Global, GracefulDegradation, HackDetect, Handoff, HeldRelease, IdempotentWrite, IdentityHandshake, IdentityMask, ImaginaryHistorian, IntentGap, InternalConsistency, InvariantFilter, LatticeCommit, LocalizedLearning, ManifestPlanning, MemeticSeed, MetaPrompt, ModestClaim, MonitorReport, MonotonicCounter, Nucleate, OntologyHandshake, OptimisticSolver, Oracle, OrchestrationLoop, OsmoticFilter, PatternEmergence, PatternSketch,

PermissionEscalate, PhasedRefinement, PromiseGraph, PromptChain, PropheticQuorum, Proprioception, ProtoPack, QuorumPulse, Realizable, RealizationProtocol, RequestFraming, Reversibility, ReversibilityCheck, Robustness, Rollout, RolloutManifest, RolloutWatch, RootHashGossip, SafetyCartographer, ShoutWhisper, SignalReflection, SimulationTrace, SolverManifest, SolverNode, SomaticMarker, SourceEvaluate, SpectralTune, StateSnapshot, StructuralCoaching, StyleSpec, SynergisticMode, Taper, ThinSlice, ThreeLevelCollision, TieredAccess, TimeboxThink, ToolDiscovery, TraceBelief, TranslationProxy, UniqueHandle, UptakeAsGround, UptakeOverTimestamp, VowOfSilence, Warmup, WorkerMode, Workflow, WorldReversible.

Infrastructure Layer (114 patterns) Operational constraints.

- **Data Structures** (80): Anomaly, Artifact, Assessment, Assumption, Audit, Ballot, Belief, Break, Card, Category, Chain, Check, CognitiveBias, ConceptAnchor, Condition, Constraint, Context, Contract, Correlation, Criteria, Critique, DAG, Datum, Event, Exception, Goal, Group, Hierarchy, Hypothesis, Identity, Ledger, MECE, MechanisticDesignProposal, Message, Meta, Metric, Mode, Nature, Observe, Option, Outcome, Overlap, Permission, Plan, Probability, Problem, ProblemSpace, Prompt, Proposal, Protocol, Prototype, Queue, Resource, Result, Risk, RuleSet, Score, ScoringFunction, Sequence, Shard, Signal, Skeleton, Snapshot, Solution, Spec, State, Step, Stream, Subject, Summary, System, Task, ToolInvoke, Topology, Transition, Tree, Value, Variable, Vector, Work.
- **Primitives** (24): Act, Actor, Aggregate, Backoff, Budget, Care, CircuitBreaker, Combine, CryptoShred, Feedback, Greet, Incongruity, Judge, Loop, NegativeProof, Probe, Quorum, Sandbox, Search, Select, StateTransition, TaskLifecycle, TimeWarpLog, Trace.
- **Verification** (10): AuditTrail, CompatibilityCheck, ExplainBeacon, HumanApprove, InputGuard, OathBind, OutputGuard, RingWitness, SpotAudit, Validate.

The four individual qualities (`Parsimony#2578`, `Novelty#2218`, `Realizable#199e`, `Expansive#ad2d`) are retained for atomic evaluation, but `PURECheck#b8ca` serves as the canonical orchestrator, a composite pattern that enforces all four properties via sequential non-compensatory evaluation using `TriGate#a30f` instances. Failure at any gate triggers immediate rejection.

References

[1] Elizabeth Gibney. Can researchers stop ai making up citations? *Nature*, 645(8081):569–570, 2025.

[2] Zifan Feng et al. SEMA: Semantic-aware monitoring and adaptation for safe LLM deployment. Unverified, 2026. UNVERIFIED: Could not confirm existence at ICLR 2026 or any repository as of March 2026.

[3] Sandro Hawke. Identification via secure definition hash. W3C Draft Proposal, 2002.

[4] Juan Benet. Ipfs - content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[5] Paul Chiusano, Rúnar Bjarnason, et al. The unison language reference, 2024. Content-addressed code architecture.

[6] Anders Rundgren, Bret Jordan, and Samuel Erdtman. RFC 8785: JSON canonicalization scheme (JCS), 2020.

[7] Dave Longley, Gregg Kellogg, and Dan Yamamoto. Rdf dataset canonicalization: A standard rdf dataset canonicalization algorithm. W3c recommendation, W3C, 2024.

[8] Joel Gustafson. Content-addressing semantic data. Knowledge Futures Group Notes, 2019.

[9] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers. In *2024 IEEE 7th international conference on multimedia information processing and retrieval (MIPR)*, pages 155–161. IEEE, 2024.

[10] Manoj Ghuhan Arivazhagan, Lan Liu, Peng Qi, Xinchi Chen, William Yang Wang, and Zhiheng Huang. Hybrid hierarchical retrieval for open-domain question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10680–10689, 2023.

[11] Erica Coppolillo and Simone Mungari. Unexpected knowledge: Auditing wikipedia and grokipedia search recommendations, 2025.

[12] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Mober, et al. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

[13] Arnav Singhvi, Omar Khattab, et al. DSPy assertions: Computational constraints for self-refining language model pipelines. In *Workshop on Systems for Next-Gen AI Paradigms (NeurIPS)*, 2024.

[14] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with ChatGPT. In *arXiv preprint arXiv:2302.11382*, 2023. Foundational work on formalizing prompt patterns using software design pattern structure.

[15] Jon Postel. RFC 761: DoD standard transmission control protocol, 1980. Be conservative in what you send, be liberal in what you accept.

[16] Henrik Westerberg. Fractal intelligence: Conceptual decomposition as problem-solving infrastructure. Zenodo, 2026. Preprint. Introduces the Fractal Intelligence architecture: recursive conceptual decomposition governed by a marginal-value rule, with a five-surface Solver Contract.

[17] Henrik Westerberg. The ontology of the alien: Escaping the median trap in LLM ideation. Zenodo, 2026. Demonstrated emergent metacognition in multi-agent ontology construction.

[18] Stewart Brand. *The Clock of the Long Now: Time and Responsibility*. Basic Books, 1999. Source of the 'Pace Layering' concept applied to civilization constraints.

[19] Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (shacl). W3c recommendation, W3C, 2017.

[20] Charles Babbage. *On the Economy of Machinery and Manufactures*. Charles Knight, London, 1832. Source of the Babbage Principle regarding the division of mental labor.

[21] Elliot Meyerson et al. Solving a million-step llm task with zero errors, 2025. Empirical validation of the Babbage Principle via 'Massively Decomposed Agentic Processes'.

[22] Google DeepMind. AlphaEvolve: A gemini-powered coding agent for designing advanced algorithms, 2025. Discovered the 48-step matrix multiplication algorithm (beating Strassen's 1969 record) and optimized global compute infrastructure.

[23] Henrik Westerberg. The superintelligence that cares about us. Zenodo, 2025. July 2025. Proposed the 'Living Taxonomy of Thought' as a prerequisite for beneficial AI.

[24] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models, 2024.

[25] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. Foundational theory for separating 'Fast Context' from 'Slow Structure'.

[26] David Wadden et al. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7534–7550, 2020.

[27] Dasha Metropolitansky and Jonathan Larson. Towards effective extraction and evaluation of factual claims. ACL 2025, 2025. Introduces Claimify, an LLM-based claim extraction method.

[28] Frédéric Berdoz, Leonardo Rugli, and Roger Wattenhofer. Can AI agents agree?, 2026. Shows LLM agents cannot reliably reach consensus even in benign settings; performance degrades with group size.

[29] Mert Cemri et al. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks*, 2025. Introduces the MAST taxonomy. Analyzed 1,600+ execution traces across 7 MAS frameworks; found 79% of failures originate from specification and coordination issues.

[30] Paula Chocron and Marco Schorlemmer. Vocabulary alignment in openly specified interactions. *Journal of Artificial Intelligence Research*, 68:69–107, 2020. Formal interaction-based vocabulary alignment for multi-agent coordination.

[31] Gottfried Wilhelm Leibniz. *Dissertatio de arte combinatoria*. Fickius, Leipzig, 1666. The foundational proposal for the 'characteristica universalis' and the derivation of logic from a namespace of prime concepts.

[32] Craig Sayers and Kave Eshghi. The case for generating URIs by hashing RDF content. Technical Report HPL-2002-216, HP Laboratories, 2002.

[33] Kurt Cagle. Why the semantic web has failed. LinkedIn, 2016. Identifies complexity, paradigm mismatch, and invisibility as barriers to adoption.

[34] Samuele Marro, Emanuele La Malfa, Jesse Wright, Guohao Li, Nigel Shadbolt, Michael Wooldridge, and Philip Torr. A scalable communication protocol for networks of large language models. *arXiv preprint arXiv:2410.11905*, 2024.

[35] Christine Lemmer-Webber. Content addressed descriptors and interfaces with Spritely Goblins. `https://dustycloud.org/tmp/interfaces.html`, 2021. Draft specification for OCapN content-addressed interface discovery.

[36] Solidity Contributors. Contract ABI specification. `https://docs.soliditylang.org/en/latest/abi-spec.html`, 2015. Defines 4-byte Keccak-256 function selectors as call routing tokens.

[37] Ian Grigg. The Ricardian contract. `https://iang.org/papers/ricardian_contract.html`, 2004. First presented 1996; formalized hash-identified human-readable contracts.

[38] Gabriella Gonzalez. Semantic integrity checks are the next generation of semantic versioning. `https://haskellforall.com/2017/11/semantic-integrity-checks-are-next`, 2017. Describes Dhall's content-addressed normal-form hashing.

[39] Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[40] Yunfan Gao et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[41] Manu Sporny, Grant Noble, Dave Longley, Daniel C. Burnett, Brent Zundel, and Kyle Den Hartog. Verifiable credentials data model v1.1. W3c recommendation, W3C, 2022.

[42] Pierre-Yves Vandenbussche, Ghislain A. Atemezing, María Poveda-Villalón, and Bernard Vatant. Linked open vocabularies (lov): A gateway to reusable semantic vocabularies on the web. *Semantic Web*, 8(3):437–452, 2017.

[43] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[44] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[45] Peter Steinberger. OpenClaw: Personal AI assistant. `https://openclaw.ai`, 2026. Open-source autonomous agent framework with multi-agent routing and persistent execution.

[46] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model context protocol (MCP), agent communication protocol (ACP), agent-to-agent protocol (A2A), and agent network protocol (ANP), 2025.

[47] Anthropic. Model context protocol. Anthropic, 2024. Open protocol for LLM-tool integration, donated to Linux Foundation December 2025.

[48] Google. Announcing the agent2agent protocol (A2A). Google Developers Blog, 2025. Open protocol for agent-to-agent collaboration, donated to Linux Foundation June 2025.

[49] Agent Network Protocol Community. Agent network protocol (ANP), 2025. Decentralized agent discovery and collaboration via DIDs and JSON-LD.

[50] Otavio Serra. Symplex: Agent semantic protocol. `https://github.com/olserra/symplex`, 2026. MCP extension using semantic intent vectors (float32 embeddings) for agent interoperability via cosine similarity.

[51] Ecma International. NLIP: Natural language interaction protocol. `https://ecma-international.org`, 2025. Ecma-430 through Ecma-434. Standardized protocol where agents use generative AI to translate between local ontologies, requiring no shared vocabulary.

[52] Charles Fleming, Vijoy Pandey, Luca Muscariello, and Ramana Kompella. A layered protocol architecture for the internet of agents, 2025.

[53] Skan AI. Agentic ontology of work (AOW): A common language for the age of intelligent automation. Skan.ai, 2026. Enterprise semantic framework defining 9 canonical entities for agent governance.

[54] Amjad Fatmi. Faramesh: A protocol-agnostic execution control plane for autonomous agent systems, 2026. Protocol-agnostic execution control plane for autonomous agent systems.

[55] Om Tailor. Verifier-bound communication for LLM agents: Certified bounds on covert signaling, 2026. Certified bounds on covert signaling in LLM agent communication.

[56] Claudiu Leoveanu-Condrei. A DbC inspired neurosymbolic layer for trustworthy agent design, 2025. Adapts Design by Contract to LLM agents; posits that agents satisfying the same contracts are functionally equivalent.

[57] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. ICSE 2026, 2026. Formal semantics for runtime rule enforcement in LLM agents.

[58] W3C WebAgents Community Group. Webagents community group report on interoperability for agents on the web, 2025. Synthesizes FIPA, BDI, and modern LLM-based agent protocols.

[59] Martin Josifoski, Lars Klein, Maxime Peyrard, Nicolas Baldwin, Yifei Li, Saibo Geng, Julian Paul Schnitzler, Yuxing Yao, Jiheng Wei, Debjit Paul, and Robert West. Flows: Building blocks of reasoning and collaborating AI, 2024. Frames AI agents as modular "semantic processors" exchanging typed messages.

[60] Microsoft. NLWeb: Bringing conversational interfaces directly to the web, 2025. Uses Schema.org as semantic foundation for agent-web interaction; every instance exposes an MCP server.