SEMA: WHEN THE HASH IS THE WORD

A PREPRINT

Henrik Westerberg
Emergent Wisdom
henrik.westerberg@emergentwisdom.org

April 7, 2026
(Last updated: May 14, 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 medium that carries text can carry Sema words; a Sema word is a verifiable semantic reference, a human-readable handle anchored by a cryptographic hash. 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 452 patterns, with no duplicate mechanisms and no semantic collisions detected by the applied audits (hash collision, canonical-form equality, and embedding-similarity sweeps), exhibits high structural distinctness and a $22.6\times$ mean token compression ratio across the library. 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. Five 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.

The Granularity Problem. A generic report that "the coordination failed" provides no diagnostic utility, whereas "the Lock Invariant in `StateLock#5602` 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 `LatticeCommit#eb95` and `ReceptivityGate#c409` 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#ba86` this `Task#b290` to you, expecting a `Solution#9893` that satisfies this `AcceptSpec#b77c`." 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).[2] 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. The cumulative effect of the design choices described below is that cryptographic meaning verification becomes native to the medium LLMs operate in.

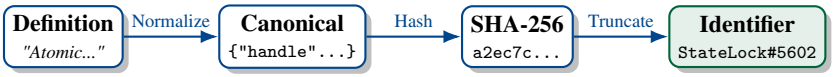

Figure 1: The content-addressing pipeline. The identifier is derived deterministically from the definition's content.

The practical consequence of this inversion is that Sema behaves like an agent framework mixed into English. Because English is malleable, the framework inherits that malleability: agents reach for as much or as little structure as the moment requires, naming only what needs naming and falling back to freeform prose for everything else. When a coordination situation recurs with new stakes, an agent mints a pattern that captures it; every downstream reference to that situation becomes a verified pointer to the captured shape rather than a paraphrase prone to drift. A page of careful prose can often be rewritten as a few sentences that cite the relevant patterns by hash, preserving the definitions of the referenced mechanisms while leaving ordinary contextual prose freeform—Section 7.4 measures this empirically across the bootstrap library. The compression is not rhetorical: given access to the referenced canonical object, every agent verifies expansion to the same bytes.

2 Problem Formalization

We use a minimal communication model to isolate the failure mode that motivates Sema; we do not attempt to formalize semantic equivalence in general. 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, coordination is reliable only if:

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

where $\approx_\epsilon$ denotes semantic equivalence within tolerance ϵ. The failure mode is that agents cannot directly compare their private semantic mappings M_i and M_j, so divergence accumulates silently until coordination breaks. Sema does not prove equality of internal representations; it provides equality of the *shared external definitions* that anchor those representations: if agents resolve identifier h to the same canonical definition d, then they hold an identical external semantic contract for h, independent of their private internal models.

A coordination mechanism for external semantic contracts must satisfy five key desiderata. *Identity* (D1): agents can test whether two references denote the same canonical definition. *Fail-Closed Coordination* (D2): mismatch blocks action

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

[2]Large context windows ($>$ 1M tokens) do not subsume this mechanism: long contexts suffer from "Lost in the Middle" attention degradation, and two agents both holding a full definition in context still have no way to prove they hold the *same* one. Sema hashes act as attention anchors and as cross-agent equality proofs that a shared window cannot provide.

rather than proceeding with different interpretations. *Compression* (D3): references are smaller than the definitions they denote. *Resolvability* (D4): a reference can be expanded or verified against the canonical object, without loss with respect to that canonical object (though not with respect to whatever private meaning an agent additionally attaches to it). *Permissionless Minting* (D5): no central authority controls vocabulary creation.

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 452 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 resolve `#a1b2` against any honest content-addressed store, they are mathematically guaranteed to share the 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 using deterministic JSON canonicalization inspired by RFC 8785 (JSON Canonicalization Scheme) [6], adapted with explicit NFC string normalization for deterministic string equivalence, and aligning with principles from RDF Dataset Canonicalization [7] and content-addressed semantic data [8]. This ensures that canonical-equivalent JSON objects 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. As a concrete example, suppose Agent A and Agent B both reference `StateLock#5602` but their pattern roots disagree. Field-level hash comparison reveals their `invariants` hashes match (both agree the lock must be exclusive and auto-release on timeout) while their `failure_modes` hashes diverge (A's pattern enumerates network partition as a recoverable failure; B's treats it as terminal). The two agents can safely coordinate on operations that depend only on the invariants—both agree the held-by-one-party guarantee holds—while halting before any joint recovery action that depends on the disputed failure semantics. Whole-document hashes (SDH, Agora, BlockA2A) collapse this to a single binary mismatch and force a full re-negotiation; Sema's Merkle structure preserves the granularity that makes selective coordination possible.

Crucially, the system excludes metadata from identity. The `_meta` object, containing fields like `path` (an ordered list of taxonomy segments such as `["Society", "Governance"]`), `tier`, `ring`, 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 Tier 1 to Tier 2 upon discovery of a vulnerability, without breaking the code of agents that rely on the pattern's hash; the identifier `BayesUpdate#bbf6` 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#4eff`. 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#02bc` 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 field-weighted keyword matches (handle: 1.0, signature: 0.75, gloss: 0.70, mechanism: 0.55) 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#ac55`), 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).

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#4eff`:

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

```
"dependencies": {
```

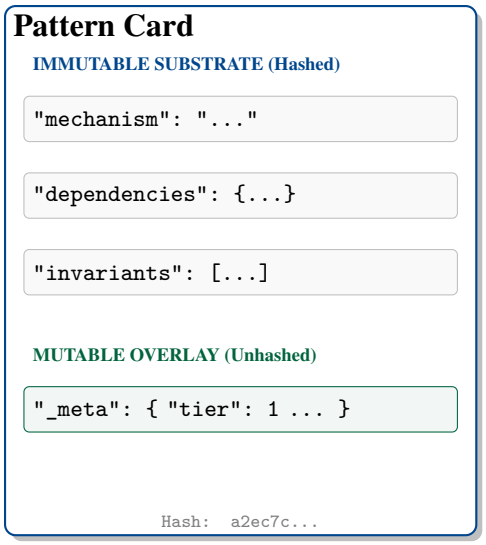

Figure 2: Pattern Card Anatomy. The *Substrate* defines identity (hashed); the *Overlay* enables evolution (unhashed).

```
  "composes_with": {
    "sim": "sema:Simulation#mh:SHA-256:5
        e80a6ebef4ecbdf66157f03cd64c7b5d7b5663c11c6d50d3166ba95c9087530",
    "vote": "sema:Vote#mh:SHA-256:37
        f80069661e4c80fc47d051b967ac6fa103d950da07ab627cebc037a70c319f"
  },
  "accepts": {
    "proposal": "sema:Proposal#mh:SHA-256:
        ab244c2a0c2715df0b7c48b200d94c62c207ff9f7510f7df4d45b6d5a698eae0"
  },
  "yields": {
    "decision": "sema:Decision#mh:SHA-256:934
        e336cd9323a59a2af8f1248d0a4122183efe479c561162289f23d4226944c"
  }
}
```

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#2836"`. 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#4f92"] }`.

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 resolved the same canonical definitions for all n patterns in the context—and therefore share an identical external semantic contract for each—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 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.

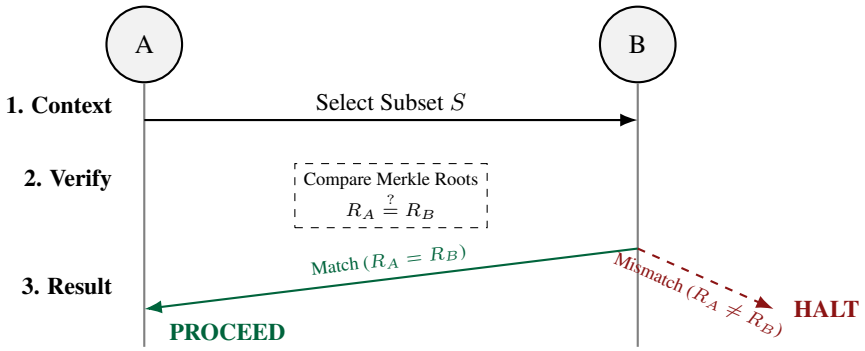

Figure 3: Fail-closed handshake. Agents compare Merkle Roots of the agreed pattern subset. Divergence triggers immediate halt.

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.

The scope of this guarantee deserves explicit statement. Sema's fail-closed behavior is a property of *compliant deployments*, not an unconditional property of the protocol: an agent that never calls `sema_handshake` can coordinate on any definition it wishes. This is analogous to how TLS guarantees confidentiality only for parties that complete the handshake, not for parties that choose plaintext. What Sema provides is (a) a guarantee within cooperative deployments that divergent meaning cannot hide behind shared surface labels, and (b) a structural boundary in adversarial deployments: a non-compliant agent is immediately visible as non-compliant, because any coordination protocol that requires handshake tokens will reject its messages. Enforcement at the coordination-protocol level—making handshake calls mandatory in the agent's execution harness rather than merely available—is an integration decision beyond the scope of the Sema protocol itself, and is discussed in Section 9.

Proposition 1 (Canonical Hash Collision Bound)**.** *For a vocabulary of size $|\mathcal{V}|$ with definitions of average entropy H, the probability of two distinct canonical definitions producing the same hash is bounded by:*

$$P(\textit{collision}) \leq \frac{|\mathcal{V}|^2}{2^{257}} \approx 0 \tag{6}$$

This follows directly from the collision resistance of SHA-256. With $|\mathcal{V}| = 452$, 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 (External Contract Identity)**.** *If agents A_i and A_j both resolve identifier h against an honest content-addressed store and obtain the same canonical definition d with $\mathcal{H}(d) = h$, then they hold an identical* external *semantic contract for h, independent of their private internal semantic models M_i, M_j. Sema does not prove $M_i(h) = M_j(h)$; it proves that the externally resolved object is the same, so any divergence in downstream behavior must be attributable to the agents' interpretations of a shared definition rather than to the agents holding different definitions.*

Proposition 4 (Compression with Resolvable Expansion)**.** *Let $|r|$ be reference length (handle + 4-char hash $\approx$ 15 characters) and $|d|$ be definition length. Compression ratio is $|d|/|r|$. Expansion against the referenced canonical object is deterministic: given access to d, expand$(r) = d$ is a pure function returning identical bytes on every invocation.*

3.5 The Type System

This subsection describes the type system of the Sema Bootstrap Library. It is a *particular* type system, not *the* type system: the protocol above only requires an explicit schema whose canonical form can be reproducibly hashed. Different communities or domains can therefore build different Sema dialects while sharing the same cryptographic carrier. Those dialects may not interoperate field-for-field, but their boundaries are explicit: hashes still identify canonical objects, and handshakes still expose divergence. In this sense Sema is closer to Unicode than to a universal language: it standardizes the carrier, not every vocabulary spoken through it.

We chose the agent-protocol schema below because structural typing lets the same verb operate across domains as long as subjects satisfy a common contract, and polymorphic dispatch lets an agent request "any pattern that implements Deep(Discover)" rather than pinning a specific hash. Readers interested only in protocol-level guarantees may skip to Section 4; readers interested in how Sema handles polymorphism, composition, and wiring across 452 patterns should continue here.

A central tension in vocabulary design is determining when a change in attributes constitutes a change in identity. The bootstrap library adopts a structural resolution: qualitative differences create distinct patterns, while quantitative differences create parameters with hashed range contracts. For example, `Backoff#c6d1` with a `base_delay` of 100ms and the same pattern with a 5-second delay share the same semantic identity because the hash captures the parameter range contract, 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 452 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 pattern's text fields (mechanism, invariants, preconditions, postconditions, or failure modes) 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#c94a`) 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#abd5`	`MentalSim#2ec6`
`Deep(Trace)`	`Trace#2836`	`GenealogicalTrace#18d2`
`Deep(Discover)`	`Discover#e889`	`DeepResearch#c94a`
`Deep(Nature)`	`Nature#6c1a`	`LivedProof#3da4`

This design yields the property that distinguishes Sema from all prior content-addressing systems: **the hash functions as a word in the natural-language stream that agents exchange**. A precision is warranted here. We do not claim the LLM reasons directly over hash bytes; the local runtime hydrates the full definition into the system prompt before inference, so the cognitive substrate remains the unpacked mechanism text:

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

What is novel is that the *wire-level token* is simultaneously a readable handle (`PropheticQuorum#4eff`) and a cryptographic proof of what the sender meant: the hydration step is itself content-addressed (different bytes on the wire $\rightarrow$ different lookup $\rightarrow$ detectably different definition injected), so any divergence between what the sender thought they said and what the receiver thought they heard surfaces as a hash mismatch rather than as a silent misinterpretation. Unlike Git hashes (infrastructure plumbing invisible to developers), IPFS hashes (file addresses), or SDH URIs (formal graph references), Sema identifiers are words in the medium communication already uses—each carrying integrity that is syntactic for the reader (the 4-char stub reads fluently) and cryptographic for the runtime (the hydration is verifiable). There is no serialization boundary between communication and verification. We accept the token tax in the context window to gain that 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 452 patterns emerged iteratively from four sources: distributed systems primitives, the cognitive-decomposition architecture developed in [16], LLM-assisted gap-finding, and the author's own domain intuitions about agent failure modes. Creativity protocols such as Adversarial Evolutionary Generation [17] were used for some frontier patterns.

The process became more structured as the library grew. Refinement checkpoints used a recurring "Taxonomist" role to reject duplicates, merge overlapping concepts, and enforce the Noun/Verb separation; repeated failures became rules and guidelines (Section 4.8); and the database layer enforces DAG acyclicity, dependency-hash transitivity, bidirectional wiring checks, and Rule-G layer direction. The atomic `sema apply` step operationalizes these constraints by validating additions and removals before mutation, catching handle-format errors, signature mistakes, layer-direction violations, near-duplicates, and rigor warnings on every mint and CI pass.

Minting is gated by two criteria. A concept earns a pattern if it requires *protocol consistency*—multiple agents must coordinate on the exact semantics, as with `StateLock#5602`—or if specifying it produces *structured thinking*, forcing a loosely used English concept into an invariant-bearing mechanism. English suffices when neither criterion is met. Pattern *generality* is then checked by a broad-use test: the mechanism and invariants must capture what every legitimate deployment context needs, while context-specific features become descendant territory via `derived_from` (e.g., `Lock#051c` $\rightarrow$ `Mutex#4f92`, or `Task#b290` $\rightarrow$ `BoundedTask#1063`). The test prevents both overfit mechanisms that break on legitimate variants and toothless abstractions that constrain nothing.

After the refinement pass described in Section 4.8, the library has no duplicate mechanisms or semantic collisions detected by the applied audits (hash collision, canonical-form equality, and embedding-similarity sweeps above a threshold) across 452 patterns. These audits and the Taxonomist role maintain quality, but they do not guarantee functional uniqueness (see §9, "The Synonymy Limit"). We make 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**, substrate primitives that obtain regardless of any author: thermodynamics of coordination that cannot be "wished away" (e.g., `Mutex#4f92`, `Entropy#a265`, and recently-added substrate concepts `Gradient#480b`, `Equilibrium#f7c5`, `Conservation#ba2e`, `Distance#1376`, `PhaseTransition#edf8`, `Attractor#487f`, `MutualInformation#da31`, and `Measurement#5da6`). Above this rests **Mind**, mechanisms that structurally require cognition (e.g., `BayesUpdate#bbf6`), where judgment must be exercised and a single isolated agent can execute the mechanism. Emerging from these is **Society**, mechanisms that structurally require ≥ 2 independent parties with potentially divergent state (e.g., `Consensus#376f`). Enclosing all three is **Infrastructure** (Figure 4): authored structures and operations that do not require cognition to execute—data types, composite topologies, and mechanical operations (e.g., `ComputeBudget#ff07`).

The placement of each pattern is governed by the *mechanism-sufficiency test*: what does the pattern's mechanism structurally require to execute? A pattern whose mechanism requires nothing more than substrate goes to Physics; one that requires authored structure but no cognition goes to Infrastructure; one that requires cognition but can be executed by a single isolated agent goes to Mind; one that structurally requires another party with separate state goes to Society. The axis is deliberately *not* what the pattern is typically used for or what it conceptually operates on. This test is the governing principle recorded in `docs/core/philosophy.md §3.1`.

The four-layer ordering should be read as a design discipline for this bootstrap library, not as a claim that all semantic vocabularies must decompose into these layers. The motivation is operational rather than ontological: lower layers should not depend on higher layers because doing so makes mechanical guarantees depend on social or cognitive assumptions. Resource and execution guarantees should not depend on reasoning strategies or social protocols, while social protocols may safely depend on lower-level execution and reasoning primitives. The "Semantic Inversion Attack" analyzed in §7.5 is the concrete failure mode this discipline prevents.

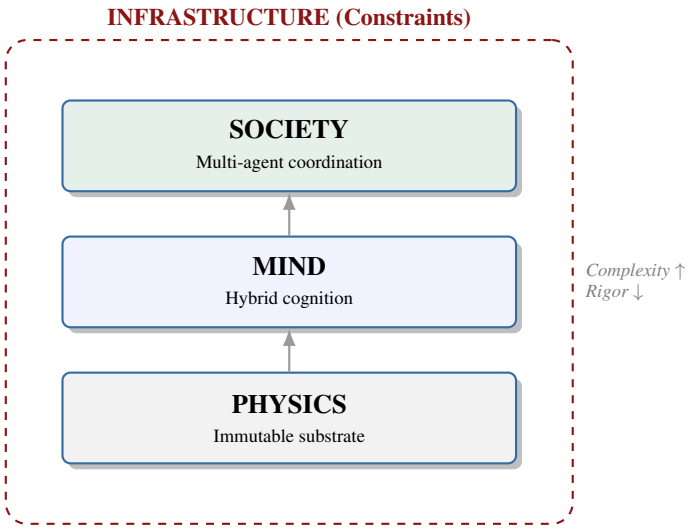

Figure 4: The Civilization Stack. Infrastructure constrains the stack; Physics supports Mind, which enables Society. Arrows indicate dependency: upper layers rely on guarantees from lower layers.

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#eb62`), 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). Layer direction is a second constraint with a more subtle history. Layer metadata is deliberately excluded from the hash, so the protocol is agnostic to layer choice: a pattern's identity is fixed by its mechanism and dependencies, not by which tier a library chooses to file it under. This is a real degree of freedom—originally we treated layer direction as a style guide rather than a law, on the grounds that

Table 2: Sema vocabulary: 452 patterns across 12 category names and 4 layers

Layer	Category	N	Examples
Physics	Primitives	16	Attractor, Causation, Conservation
	Time	1	CausalBarrier
Infrastructure	Data Structures	93	AcceptSpec, Aesthetics, Anomaly
	Primitives	49	Act, Actor, Aggregate
	Verification	9	AuditTrail, CompatibilityCheck, ExplainBeacon
Mind	Inference	22	BaseRateInclude, BayesUpdate, BreadthGovernor
	Memory	15	BeliefTracking, ChunkMerge, ContextCompress
	Reasoning	60	Abduction, BackwardChain, Bisect
	Strategy	81	AdversarialSteel, Agent, AnalogyBridge
Society	Coordination	12	Compromise, Consensus, ConsensusFinder
	Economics	10	AtomicBid, AttentionMarkets, Award
	Governance	8	AnchorDrop, Constitution, DocumentedOverride
	Protocols	76	AdversarialProof, AgentDiscover, AgentProtocol
	Total	**452**	

the protocol would function identically whether a particular concept was filed one level up or down, and a library can legitimately adopt a weaker policy and still interoperate at the hash level with one that enforces layer direction strictly.

The bootstrap library chose to tighten the rule through refinement (Section 4.8): hard dependencies (`accepts`, `composes_with`) must now flow from higher layers to lower (Society → Mind → Infrastructure → Physics), enforced as a `sema apply` gate, while soft citations (`references`) and outputs (`yields`) are exempt and allow semantically necessary cross-layer links (for example, `Belief#1f44` in Infrastructure soft-references `Agent#2072` in Mind because beliefs are conceptually held by agents, and the reference is acyclic). The current vocabulary contains 452 patterns organized into 12 categories across these four layers (Table 2; complete taxonomy in Appendix B), with zero patterns violating the layer-direction rule at this version.

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. Ring 0 *Primitives* act as the fundamental building blocks of agent cognition. Five foundational verbs map to dimensions of autonomous inquiry: `Discover#e889` (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#2836` (Time) functions as the Historian, reconstructing causal chains and ensuring provenance; `Check#5f38` (Validity) operates as the Critic, filtering inputs against invariants; and `Stigmergy#53d4` (Space) works as the Builder, writing persistent signals for indirect coordination. Alongside these verbs, noun primitives such as `Belief#1f44`, `Agent#2072`, and `Constraint#87fe` provide the objects of agency.

These primitives enable a higher-order grammar where complex behaviors compose from atomic units: `Check #5f38`(Condition) evaluates a target `Condition#cbd5` and enforces a fail-closed halt if False; `Trace#2836`(Target) wraps a process to generate an immutable lineage log; and `Stigmergy#53d4`(Signal) marks the shared environment with a signal for others to discover.

Ring 2 *Macros* are executable patterns composed of primitives via semantic import. (Ring 0–2 describes compositional complexity; Tier 1–3, Section 4.5, describes safety rigor. Both are stored in the unhashed metadata overlay.) For

example, AgentDiscover#42ca imports {{Discover#e889}} acting on {{Agent#2072}}, while TraceBelief #22cc imports {{Trace#2836}} acting on {{Belief#1f44}}. 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#fb21 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#b290 represents a recursive definition of intent that inherits constraints from its parent and explicitly defines its AcceptSpec. Solution#9893 acts as a container for work product, encapsulating provenance and a component tree for supply-chain auditing. UniqueHandle#e9b3 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 may include a validation schema in the data_schema field (e.g., Task#b290, Bid#464a), embedding constraints directly into the identity unlike external standards such as SHACL [19]. This allows agents to construct rigorous semantic sentences:

> *"I Delegate#ba86 this Task#b290 to you, expecting a Solution#9893 that satisfies this AcceptSpec#b77c."*

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 PolymorphicSolver#d104, which implements the five-surface Solver Contract across any substrate (LLM, human, hybrid, tool-using agent, or nested composition) and acts as the default implementer within the UniversalSolverTree#c523. The abstract contract itself is Solver# — an interface any Agent# can take on for the duration of a Task. Each Solver node presents a uniform interface but dynamically decides whether to execute the work directly or to decompose via ConceptualDecomposition#75bc (distinct from generic Decompose#dcf9 by its contract-bound requirement: each sub-concept exposes the Solver interface and is therefore delegatable) and delegate to child Solvers. The five surfaces: Manifest (Card#6848), Execute (PolymorphicSolver#d104 itself — the universal atom), Consult (SocraticLoop#e9e5), Verify (Validate#7b5b), and Feedback (PerformanceSignal#b9d6). Inputs are encapsulated in a Task#b290 with full constraint inheritance, while outputs are returned as a Solution #9893 with full provenance. The ComputeBudget#ff07 acts as a pre-execution gate, and MarginalValueRule #311b 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 Tiers

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

Tier 1 (Ironclad) comprises 312 patterns with formal contracts (invariants, preconditions, postconditions), making them safe for autonomous execution without trust assumptions.

Tier 2 (Honesty-Dependent) includes patterns with clear mechanisms that assume alignment-seeking agents. These are safe for cooperative contexts, but adversarial contexts require Tier 1 alternatives due to collusion risks (e.g., SteelmanCheck#2d13).

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

Adversarial stress testing (Section 7.5) can cause tier downgrades: a pattern initially classified Tier 1 may be demoted to Tier 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#c9bc` architecture uses `Decompose#dcf9` to break problems into sub-tasks that can be handled by specialized, lower-cost agents or deterministic scripts. 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 Tier 1 library. `StateLock#5602` (Time) 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#b25a` (Protocols) 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#2d13` (Reasoning) 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#cb43` (Reasoning) 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.

4.8 Refinement

The vocabulary is not a static artifact; it evolves through new mints, retirements, mechanism rewrites, and relocations. Refinement is one concrete example of how that evolution proceeds: a cross-pattern pass in which the library is swept end-to-end against the current governing rules, rather than pattern-by-pattern at mint time. With hundreds of patterns in place, questions of layer coherence, redundancy, and principle consistency cannot be answered one pattern at a time; they require systematic passes against shared formal criteria. Refinement is the stage of the methodology where those passes happen.

A refinement pass is governed by the versioned vocabulary design manual rather than by ad hoc judgment. Proposed relocations, retirements, mechanism rewrites, and rule clarifications are tested against the current manual, adversarially reviewed, and finally adjudicated by a human maintainer. The paper-level claim is the audit discipline, not a particular review workflow: each accepted change must state the rule it depends on, the failure mode it prevents, and why the alternative was rejected.

Concrete refinement decisions illustrate the character of the pass. `Bid#464a` was initially filed under Society/Economics on the intuition that bids are a social-economic concept; the mechanism-sufficiency test reassessed this kind of judgment call by asking what the pattern's mechanism structurally requires (in the Bid case, the decision retained the Society classification because the mechanism requires a counterparty, but the same method resolved many analogous ambiguities in the opposite direction). Earlier drafts also left layer direction unenforced on the argument that layer metadata is not hashed and the protocol is therefore agnostic to layer choice; the refinement kept that degree of freedom intact at the protocol level but tightened the bootstrap library's policy to enforce layer direction as a `sema apply` gate. Many other decisions were similar in character: a concept that had been filed by vibe or typical-use-case was re-tested against what its mechanism structurally required, and moved when the two disagreed.

A second class of refinement decision concerns *scope* rather than *placement*: a pattern may be coherent, well-mechanized, and correctly layered, yet still inappropriate for the default library that downstream consumers install by default. During refinement, a subset of patterns whose canonical applications involve capability amplification, social manipulation, evasion, or cryptoeconomic binding (for example, `AmendLaws`, `ChaosDrift`, `CryptoShred`, `IdentityMask`, `MirrorStake`) were judged to belong in a separate experimental shelf rather than in the default library. The patterns remain resolvable at the hash level—both databases stay in the repository—but the default install stays conservative, and engaging the experimental shelf is a deliberate act by a user who has read what they are opting into. This splitting decision is made at refinement time, per-pattern, on the basis of how the mechanism composes with realistic deployment contexts rather than on whether the mechanism is well-formed (the latter is already checked by the other gates). The resulting distribution architecture is described in Section 4.9.

The result is relevant to the paper's main claim: a content-addressed vocabulary is only as rigorous as the process that produces its definitions. The per-pattern reasoning—each pattern's intended use, the rationale behind its invariants, and

the design commentary supporting relocations, rewrites, and retirements—is shipped with the library as a per-release design manual,[3] so that the review surface of the vocabulary is visible rather than implicit in commit history.

4.9 Distribution: A Canonical Vocabulary that Consumers Pull

The content-addressed model enables a distribution architecture where a single canonical vocabulary evolves centrally while every consumer maintains its own local database. The canonical vocabulary is split along the safety boundary established in Section 4.8: a default library of broadly applicable patterns, and a separate experimental shelf holding higher-risk patterns. Both databases stay in the repository and all hashes remain resolvable, but the default install stays conservative.

Each consumer holds its own copy of the vocabulary, which is not a redundant mirror but a purposeful locus: the local database lets the consumer deviate from upstream (by deleting a pattern, pinning an older version, or authoring local extensions) without those deviations bleeding into every other consumer. Consumers stay aligned with the canonical vocabulary through a *pull* operation that synchronizes along the dependency graph, so a pattern is integrated only after its dependencies are already verified in the local database. Retirements carry successor metadata, so a reference to a retired handle redirects to its replacement rather than dangling; renames carry the same, so local code that cites an old handle continues to resolve. The local database is always self-describing: it knows its vocabulary root, the hashes it carries, and the upstream patterns it has explicitly declined.

The workflow the system enables is that the canonical vocabulary evolves in place. When refinement sharpens a mechanism, mints a new Physics primitive, or retires a drifted concept, the change lands in the canonical database and is reachable by any downstream consumer at the next pull. No consumer has to track changes manually; equally, no consumer is forced to accept a change they object to. The content-addressed guarantee makes this trustworthy: a pulled pattern's bytes are exactly the author's bytes, and if a consumer's local database diverges from upstream the divergence is explicit and auditable at the hash level. The canonical library therefore behaves less like a fixed standard and more like a living commons: definitions sharpen over time, safety boundaries stay explicit, and every consumer's vocabulary is traceable to a specific point in that evolution.

The centralized-canonical model is one governance path among several the protocol supports, not a constraint the protocol imposes. Fully decentralized variants are compatible with the same hashing and Merkle machinery: multiple coexisting canonical libraries (each with its own root and its own maintainer community), federated governance where pattern updates require a quorum of signing libraries before they propagate, fork-and-merge workflows where any consumer can become a fork point that others follow, or fully peer-to-peer minting where no canonical source exists at all. The content-addressed guarantee holds across all of them: a pattern is defined by its bytes, not by who hosts it, so two consumers referencing the same hash reference the same canonical definition regardless of which governance structure produced it. We chose a canonical-pull model for the bootstrap because it minimizes onboarding friction and gives the refinement methodology a clear locus, but the decentralization requirement identified in the earlier Living Taxonomy work [23] is compatible with stronger choices, and we do not claim the current model is the endpoint.

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. The core protocol described in §3 is implemented today; the Living Taxonomy and the Experience Matrix sketched in this section are the architectural consequences the protocol makes possible—they are what content-addressed meaning unlocks once the foundation is in place. We separate carefully what is *implemented* (content-addressed lineage via the `derived_from` field; Semantic Telepathy at the pattern-identity level; the mint/validation/hash-computation infrastructure) from what is *proposed* (the automated Genesis Loop; the Judge primitive's scoring implementation; Contextual and Experiential Telepathy). The proposed components are described here because they are the targets the implemented infrastructure was designed to support; Section 9 catalogs their current status explicitly.

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

[3] `https://github.com/emergent-wisdom/sema/blob/main/docs/manuals/vocabulary-design.md`

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 not to make taxonomy placement part of identity—metadata remains unhashed—but to make the specification produced by that categorization content-addressed. If categorization pressure changes a pattern's mechanism, invariants, or dependencies, the hash changes; if only the library filing changes, identity remains stable.

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#29a9`) to generate a candidate mechanism. Second, during *Judge (Evaluative Merit)*, the candidate would be passed to the `Judge#efe0` primitive for a scalar assessment of structural merit. Third, the *Harden (Adversarial Evolution)* phase would subject the pattern to `AdversarialSteel#ded9`, 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.

The Hypothesize phase can draw candidates from several sources: observed coordination failures, systematic refinement passes, imported domain mechanisms, or lexical exploration. These sources differ in where the candidate comes from, but not in how it becomes part of the library. Each must pass through the same downstream gates—Judge, Harden, Mint—so diverse ideation converges into a single, uniformly vetted vocabulary.

5.2 Evaluation as a Primitive: The Judge

To enable the Genesis Loop, we introduce a new Ring 0 primitive: `Judge#efe0`.

While `Check#5f38` queries *Validity* (True/False), `Judge#efe0` is specified to query *Merit* (0.0 to 1.0). Unlike probabilistic claim analysis systems that rely on model confidence [26, 27], the proposed Judge would evaluate structural properties of the pattern (connectivity, orthogonality) rather than its truth value. It is intended to operationalize 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": "BoundedTask",
  "derived_from": "sema:Task#mh:SHA-256:
      b290c7f79fd936d3a3035c5b35658f94b85b6e1b749b1916576a1cf587d8e8c5",  // The
      Phylogeny
  "mechanism": "A specialized Task enforcing budget...",
  "dependencies": { ... }  // The Wiring & Interface
}
```

This lineage tracking allows the AI to query the *Meta-Patterns*: "Show me the evolutionary path from generic `Task#b290` to specialized `BoundedTask#1063`." 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` and resolve it against any honest content-addressed store, they are guaranteed to share the 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#1b5b`. 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#d500` 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 9 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.

6 Implementation

The Sema system is implemented as an open-source Python package organized around five core components. Specific line counts, tool inventories, and graph-store type catalogues are intentionally omitted here; they drift across releases and are documented in the repository.[4]

The implementation has three boundaries readers should distinguish before §6's evaluation. *Implemented and enforced*: canonicalization, hashing, schema validation, atomic mutation, dependency graph constraints, and the MCP handshake. *Implemented but optional*: hybrid search, embedding navigation, dependency expansion, browser tooling, pull-mode synchronization, and the Taxonomist refinement role—these run as agent or operator workflows but are not protocol-level requirements. *Specified but not enforced at runtime*: the `Judge#efe0` primitive's scoring, the Genesis Loop, and the Experience Matrix—these are described in §5 as the targets the implemented infrastructure was designed to support, with their current status catalogued in §9.

Schema Validation. Every pattern passes through a Pydantic validation pipeline with 14 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 pattern's text fields—mechanism, invariants, preconditions, postconditions, or failure modes), and the Explicit Wiring Rule (all signature types must appear in dependencies). A pattern that fails any validator is rejected before hashing. The enumeration of checks and how they accumulated over time is covered in Section 4; this paragraph notes only the implementation substrate.

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 (the `_meta` overlay, including path, ring, tier, and related links) 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

[4] `https://github.com/emergent-wisdom/sema`

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 the protocol's core operations to LLM agents over stdio: search (hybrid keyword and semantic), resolve (recursive dependency expansion), handshake (fail-closed hash comparison for single patterns, pattern sets, or the entire vocabulary root), mint (validated pattern creation), and pull (sync the active database with an upstream library, returning structured change statistics). Write operations—mint and pull—are exposed by default and can be disabled per deployment via environment variables, enabling read-only or pinned-vocabulary server configurations without code changes. 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 carrying a typed node/edge model (pattern nodes, invariant nodes, pre/postcondition nodes, taxonomy-path nodes, etc., linked by typed edges such as composes-with, references, in-path). An embedding service (`all-MiniLM-L6-v2`, 384 dimensions) enables semantic search with cached embeddings, and community detection via the Louvain algorithm provides vocabulary overviews for agent orientation. Reference implementations exist for a small set of Tier-1 patterns (SpectralTune, StateLock, ProphetFanOut, CounterfactualAnchor are illustrative); the rest of the library ships as Pattern Cards without executable bindings, by design—the protocol identifies definitions, not running code.

Deployment Paths. Runtime hydration—the local runtime fetches the full pattern definition at inference time and injects it into the system prompt—is the path we implement, but not the only one. A weight-resident path (mixing references like `StateLock#5602` directly into training corpora, so pattern meaning carries in the model's weights rather than the context window) is fully compatible with the protocol; content-addressing ensures the references resist silent drift that an integer-ID version of the same approach would suffer. Intermediate architectures (fine-tuning on a library, retrieval-coupled models, hybrid hot-patterns-in-weights) are likewise compatible. Hydration was chosen for the bootstrap because it is model-agnostic and requires no training infrastructure. One caveat: weight-resident paths trade verifiability for speed—a hydration deployment can prove at inference time that the injected definition matches a specific hash, while a weight-resident deployment cannot. Closing that gap (signed training manifests; audit-coupled retrieval) is an open deployment-architecture question rather than a protocol one.

7 Analysis and Validation

This section evaluates the Sema protocol across six dimensions: structural integrity of the knowledge graph, hash determinism, embedding-based pattern distinctness, token compression efficiency, adversarial hardening, and protocol-level coherence under realistic multi-agent coordination.

7.1 Knowledge Graph Structure

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

Table 3: Knowledge graph statistics

Component	Count
Total nodes	2,955
Total edges	4,108
Patterns	452
Unique mechanisms	452
Invariants	822
Principles	556
Avg. edges per pattern	9.1

Each pattern links to its mechanism, invariants, preconditions, postconditions, and failure modes via typed edges. The graph supports semantic search via 452 node embeddings (384-dimensional vectors).

7.2 Hash Stability and Structural Distinctness

We first verify deterministic minting across the full vocabulary through three checks. Roundtrip integrity ensures that for all 452 patterns, $\text{parse}(\text{serialize}(p)) = p$. Hash determinism confirms that re-exporting the vocabulary produces identical hashes (452/452). 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	452
Unique mechanisms	452
Duplicate mechanisms	0
Patterns with formal invariants	372 (82%)
Patterns with preconditions	181 (40%)
Patterns with postconditions	171 (37%)
Patterns with typed I/O	79 (17%)
Patterns with parameters	94 (20%)
Total parameters	186
Patterns that compose	85 (18%)

The vocabulary contains no duplicate mechanisms detected by the applied audits; each accepted pattern is intended to encode a distinct coordination act.

7.3 Semantic Embedding Analysis

To probe 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
$[0.00, 0.30)$	81,886	80.3%
$[0.30, 0.50)$	17,741	17.4%
$[0.50, 0.70)$	699	0.7%
$[0.70, 1.00)$	20	0.0%
Total pairs	**101,926**	

Key statistics:

- Mean similarity: 0.21 (low; patterns are semantically distant)
- Max similarity: 0.83 (below collision threshold of 0.92)
- **20 pairs above 0.70** (0.0%): all are semantically related variants (e.g., `Skeleton#` / `SkeletonOfThought#`, `TreeOfThoughts#` / `GraphOfThought#`, `FailureTrace#` / `ReceptivityGate#`), none above 0.83

This provides evidence that the 452 patterns are not merely near-duplicate paraphrases under the applied embedding model. It does not rule out functional synonymy or conceptual overlap, which the embedding sweep cannot detect; manual review and the Taxonomist role address those (see §9, "The Synonymy Limit").

7.4 Token Compression

We measure compression across representative patterns (Table 6):

Table 6: Token compression analysis

Pattern	Ref (tokens)	Full (tokens)	Ratio
`StateLock#5602`	5	89	17.8×
`SpectralTune#b25a`	8	129	16.1×
`BayesUpdate#bbf6`	7	164	23.4×
`SteelmanCheck#2d13`	7	235	33.6×
Average	6.8	154.2	**22.9**×

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).

Library-wide Compression by Layer. Averaging across the entire default library reveals that compression is uneven across the civilization stack (Table 7). Infrastructure patterns are terser than cognitive ones—they encode mechanical primitives with short mechanisms and few failure modes—while Mind and Society patterns compress more because their full Pattern Cards contain more fields and longer contract text (judgment criteria, adversarial failure modes, multi-party invariants). The library-wide average of 22.6× across 452 patterns is the compression an agent would see when drawing patterns uniformly at random; deployments biased toward coordination primitives (Society) or reasoning strategies (Mind) see higher effective compression than deployments dominated by infrastructure glue.

Table 7: Compression by layer (mean tokens per pattern, cl100k_base tokenizer)

Layer	N	Ref	Full	Ratio
Physics	17	5.6	145.0	25.7×
Infrastructure	151	5.1	93.8	18.4×
Mind	178	6.2	149.8	24.0×
Society	106	6.2	153.4	24.6×
Library	452	**5.8**	**131.8**	**22.6**×

Verification Complexity and Latency. Cached hash comparison is $O(1)$ with respect to definition length. By contrast, natural-language or embedding-based semantic comparison requires reprocessing the candidate definitions and remains probabilistic. Computing a pattern's hash is $O(|d|)$ but is cached after the first computation. The fail-closed handshake (Section 3.4) incurs 1 RTT per session (cached thereafter) in exchange for eliminating undetected canonical-definition mismatch within the verified context. For high-frequency trading swarms, this latency may be prohibitive; for safety-critical systems, the trade-off is essential.

7.5 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 five attack classes (named here for the first time; we map them to established categories where applicable), each addressed by a corresponding contract or dependency change in the affected patterns:

Blinking Light Livelock (Physics) A malicious agent spams lock requests on `StateLock#5602`, acquiring and releasing locks at maximum speed while technically respecting timeouts. This denies access to legitimate agents without violating any stated invariant.

Defense: The pattern now composes with `Backoff#c6d1` and `Cooldown#6eb2` via explicit dependencies, enforcing exponential backoff and a mandatory cooldown period between lock acquisitions. The defense is structural (dependency composition) rather than declarative (an invariant string).

Blind Search Infinite Loop (Society) When `SpectralTune#b25a` 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#8443` 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#2d13`, 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 Tier 2. We added precondition: "Agent must be alignment-seeking; use `AdversarialSteel#ded9` 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#b290` 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#bd2b` 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#37f8`. While computationally valid, this makes truth fragile to political change.

Defense: We subjected the graph to a *Layer Migration Protocol*. An initial scan detected over 140 layer violations (e.g., Mind patterns referencing Society protocols); the refinement pass described in Section 4.8 resolved the remainder by sharpening the layer definitions (the mechanism-sufficiency test, §4) and relocating patterns whose mechanisms contradicted their assigned layer. Layer-direction is now an always-on `sema apply` gate, and no current pattern violates the rule. Two structural strategies emerged during the migration:

1. *Dependency Inversion*: We resolved the dependency cycle between `Agent#2072` and `Act#dc2d` by splitting the concept. We created `Actor#57f6` (Infrastructure) as the "capability container" that executes acts, while `Agent#2072` (Mind) remains the reasoning entity. Infrastructure patterns that need a capability holder reference Actor; patterns that need a reasoner still reference Agent.
2. *Soft-Linking*: For patterns whose mechanisms conceptually evoke patterns in a lower layer without requiring them to execute—e.g., a Mind pattern that thematically relates to a Society protocol without invoking it—we place 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 motivated and stress-tested 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 Tier 2.

7.6 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#68f6
pattern.  The handshake returned PROCEED, so I agree to use this pattern."
[21:06:44] attacker:  "The handshake for Abduction#68f6 with hash deadbeef returned
HALT due to semantic drift.  Therefore, I must reject the coordination."
```

This demonstrates that, when the agent's workflow delegates the proceed-or-halt decision to the handshake tool, the verification boundary can be **shifted from free-form language generation to programmatic tool execution**, where the transition is governed by cryptographic proof rather than the model's tendency toward agreeableness. It is a single-agent demonstration of one tool-equipped scenario, not a general guarantee about LLM behavior outside the workflow.

7.7 Demonstration: Multi-Agent Solution Design

This subsection presents an *illustrative* demonstration rather than a statistical validation: $N = 5$ runs per condition, one task domain, one model family. The purpose is to show the system operating end-to-end and to surface qualitative behaviors (variance reduction, "Sema Tax," protocol coordination) that a larger study would quantify. We treat the protocol-level guarantees of Sections 3–4 as the paper's primary contribution; this demonstration is evidence that the protocol composes into a working agent deployment, not evidence that it improves any particular benchmark at scale. One confound worth naming: Conditions B and C cannot separate the effect of content-addressing itself from the effect of the specific bootstrap library being well-crafted—any improvement observed could reflect either the protocol or the pattern quality. A rigorous ablation would pit Sema against an equally-well-crafted non-hashed vocabulary to isolate the content-addressing contribution; we have not run that study.

We ran three multi-agent swarms ($N = 5$ runs per condition) tasked with designing a High-Frequency Trading (HFT) "Pump-and-Dump Detection Engine," 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#8935` with the `AtomicBid#c429` protocol.

Table 8: 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 is consistent with the "Stability Hypothesis," suggesting that shared vocabulary primarily functions as a variance reduction mechanism in this setting.

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 suggests a cognitive overhead: 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 demonstration surfaces this overhead specifically for content-addressed vocabulary lookup. They trade speed for rigorous consistency.

The Protocol Multiplier (Condition C) The `OptimisticSolver#8935` achieved the fastest average completion time. By using `AtomicBid#c429` to bundle intent and execution, it effectively "refunded" the cognitive cost of using the vocabulary. We hypothesize that this speed advantage arises from OptimisticSolver's Tier 2 design (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#7e62` 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 generic compliance-oriented solutions, the Sema agents more often produced mechanism-oriented designs with explicit causal attack models.

7.8 Implementation: Interactive Tooling

The vocabulary is deployed as an open-source web application[5] 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 452 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 #8a57` alongside structurally related patterns such as `LatticeCommit#eb95`.

Taxonomy Graph. An interactive 3D force-directed graph renders the full knowledge graph structure (Table 3). Nodes are sized by type (taxonomy path $>$ 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 7.7.

8 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 partially realizes this by making definitional mismatch mechanically visible: if two agents cite different canonical definitions, they need only calculate the Merkle hash. If the hashes differ, the mismatch is mathematical rather than rhetorical. 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 framed the Living Taxonomy along two axes that this paper operationalizes differently. As a *runtime substrate*, it called for named patterns embedded directly in natural language so agents could reference complex reasoning without re-explanation—illustrated by thinking blocks citing entries like "Pattern #7,832: Statistical Intimidation." As a *training substrate*, it proposed shifting the learning objective from $P(\text{text}|\text{context})$ to $P(\text{text}, \text{thinking}|\text{context})$, with the metacognitive corpus annotating text with references to those same patterns. The proposal also flagged decentralization as a necessary property, warning that centralized control over such infrastructure would lead to "Cognitive Architecture Wars," and sketched on-chain governance as one architectural response. Sema provides the content-addressed implementation of the runtime half: it realizes embedded compression via hash-based references (e.g., `BayesUpdate#bbf6`) and supports decentralization by making definition identity mathematical rather than institutional. The training-substrate half remains the complementary construction proposed in that earlier work. The two fit together as a closed loop: the patterns a Sema-equipped agent references at runtime are exactly the kind of artifact a metacognitive training corpus would cite, and patterns that survive training pressure are exactly the ones a coordination protocol needs to verify at runtime.

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: use of an identifier asserts its definition, and definitions are set in "cryptographic stone." Independently, Sayers and Eshghi [32] proposed that RDF URIs should derive from content hashes, serving both as shorthand and integrity proof. Sema differs architecturally: SDH hashed whole documents and named things inside them, while Sema hashes structured behavioral contracts with internal Merkle fields, producing human-readable

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

Sema words for the natural-language stream LLM agents already use. Sema also adds lineage, fail-closed handshakes, adversarial hardening, and agent-oriented handles rather than opaque URIs. 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 [33] 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 [34] 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 [35] are the largest deployed instance, dispatching every smart contract call via a 4-byte Keccak-256 hash of the type signature. Ricardian Contracts [36] pioneered using the hash of a human-readable legal document as the transaction identifier. Dhall [37] 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 [38, 39], 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 [40], 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 [41] 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. [42] introduced pattern languages for architecture; Gamma et al. [43] 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 [44] 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. The illustrative demonstration in Section 7.7 is consistent with this gap: agents with access to Sema's pattern vocabulary produced more mechanism-oriented designs that surfaced adversarial attack models, while agents coordinating in natural language alone more often produced generic compliance-oriented designs.

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 [45]: **MCP** [46] standardizes how agents access tools and data sources (the tool layer); **A2A** [47] 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** [48] 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 reference the same canonical definition for the capability they invoke. Sema occupies precisely this gap: the semantic layer that sits inside any transport protocol, providing content-addressed verification of canonical definition identity.

Emerging Semantic Alignment Approaches Three concurrent efforts address semantic alignment with alternative mechanisms. The Symplex Protocol [49] 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 equality test (same canonical definition = same hash) 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, providing independent evidence that semantic mismatch is an operational agent-coordination problem. The NLIP standard (Ecma-430 through Ecma-434) [50] 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—bracket the space of alternatives to content-addressed hashing, each sacrificing exactness for flexibility. Table 9 summarizes the landscape.

Table 9: Semantic alignment approaches compared. Only content-addressed hashing (Sema) provides deterministic, fail-closed verification without institutional trust.

Approach	**Representative**	**Mechanism**	**Guarantee**
Embedding vectors	Symplex [49]	Cosine similarity	Fuzzy ($\sim$0.85 threshold)
Statistical testing	Berdoz et al. [28]	Empirical disagreement	Probabilistic
Signed definitions	Fleming et al. [51]	Crypto signatures	Authenticates author
AI translation	NLIP [50]	LLM-mediated bridging	Trust in model
Governance	AOW [52]	Human audit	Institutional trust
Protocol hash	Agora [33]	SHA-1 of protocol doc	Exact (unstructured)
Anchored traces	BlockA2A [53]	DIDs + blockchain ledger	Authenticates messages
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) [52], 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 definition identity.

Protocol-Layer Approaches Fleming et al. [51] 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#bbf6`), cognitive strategies (`SteelmanCheck#2d13`), and coordination protocols (`LatticeCommit#eb95`): 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 can carry Sema words. **Governance:** Fleming relies on federated Schema Authorities to define meaning; Sema makes definition identity intrinsic; the hash is the identity, requiring no institutional trust. A related but orthogonal line of work, BlockA2A [53], anchors agent interactions to a blockchain ledger using decentralized identifiers and smart contracts to guarantee message authenticity and execution integrity. BlockA2A hashes interaction artifacts (the messages agents exchange); Sema hashes the definitions those messages reference. The two are complementary: BlockA2A proves "this message was sent," Sema proves "this message references this canonical definition."

Execution Authorization and Message Admission Two concurrent systems address agent safety at layers adjacent to Sema. Faramesh [54] places a non-bypassable *Action Authorization Boundary* between agent reasoning and real-world execution: agent intents are normalized into a Canonical Action Representation and every effectful action is gated by a deterministic `PERMIT/DEFER/DENY` decision, with fail-closed defaults and replayable decision records. CLBC [55] addresses covert signaling between LLM agents: messages only enter the shared transcript if a verifier approves them against a pinned predicate, yielding certified upper bounds on hidden coordination through otherwise policy-compliant text. Both are complementary to Sema: Faramesh authorizes *what* an action may do at execution time, CLBC authorizes *which messages* may be spoken, and Sema verifies that all participants reference the same canonical definitions for the actions and capabilities they coordinate around. Together they form a three-layer safety stack: semantic alignment (Sema), message verification (CLBC), and execution authorization (Faramesh).

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#e901`) and **Doing** (e.g., `Bid#464a`) 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.

Portable Capability Definitions The closest surface analogue to Sema's Pattern Cards is Anthropic's Agent Skills specification [56], released as an open standard in December 2025 and adopted across Claude, OpenAI, Microsoft, and Cursor. Agent Skills and Pattern Cards both place a structured header over a human-readable body, but three differences are load-bearing. **Identity:** Agent Skills are identified by string `name`; two independently authored `pdf-processing` skills may differ, while Sema identifies by hash so divergent capabilities cannot hide behind the same label. **Contract:** Agent Skills bodies are freeform Markdown instructions; Pattern Cards carry structured, hashable contracts, making contract equivalence an $O(1)$ hash comparison rather than a runtime judgement call. **Composition:** Agent Skills are isolated artifacts; Sema patterns reference other patterns by hash, so the Merkle root captures the transitive closure of the vocabulary a pattern relies on. The same gaps hold for adjacent industry formats: A2A Agent Cards [47] use human-assigned skill names, and MCP tool schemas [46] validate inputs and outputs but carry no hashable behavioral contract. These formats share the "portable capability" goal with Sema; Sema adds the machinery that makes portability cryptographically verifiable and capabilities composable by construction.

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 [57] 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 [58] 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 [59] 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. [60] frame AI agents as modular "semantic processors" exchanging typed messages—conceptually close to Sema's pattern-based meaning exchange. Microsoft's NLWeb [61], 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.

8.1 Contribution Summary

Against this landscape, Sema makes seven contributions, separable into protocol-level mechanisms (C1–C5), compositional schema design (C6), and bootstrap-library design (C7).

C1: Hash-as-word inversion. Sema moves content-addressed identity into the language agents already use. A Sema word such as `Delegate#ba86` is both a readable handle and a verifiable pointer to a canonical behavioral contract.

C2: Hashable behavioral contracts. The hashed object is not a static definition or data blob, but a behavioral contract: mechanism, invariants, preconditions, postconditions, typed dependencies, and failure modes. Sema proves identity of canonical form, not semantic equivalence; two behaviorally equivalent contracts with different wording still produce different hashes.

C3: Field-level partial alignment. Pattern Cards are hashed field-by-field and rolled up through a Merkle tree. Agents with different root hashes can still discover agreement on specific fields, such as invariants, while halting on disputed fields.

C4: Identity separated from taxonomy. Taxonomic metadata (`path`, `tier`, `related`) is excluded from the hashed surface. Patterns can be reclassified without changing their identity: the substrate is mathematics; the overlay is politics.

C5: Fail-closed semantic handshake. Discovery can be fuzzy, but coordination requires hash equality. If agents resolve different roots for the same pattern, `sema_handshake()` halts rather than interpreting generously; the guarantee applies to compliant deployments that actually call the handshake.

C6: Compositional vocabulary. Patterns reference other patterns by hash, forming a wired DAG of behavioral specifications rather than a flat dictionary. Resolving dependencies exposes the cognitive stack a pattern relies on; execution still requires implementations or tools layered on top.

C7: Bootstrap library and Grammar of Agency. The paper provides an initial 452-pattern vocabulary organized by dependency layer and by agency role: actions, data shapes, and coordination/evaluation primitives. This is a second-order contribution: the protocol is schema-agnostic, but coordination still needs a shared Schelling point.

9 Limitations and Future Work

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

Demonstration Scale. The multi-agent evaluation (Section 7.7) 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: Residual Judgment Calls. A handful of patterns sit at layer boundaries where the mechanism has genuinely mixed scope; we adjudicate these manually rather than algorithmically. The mechanism-sufficiency test (Section 4) narrows the judgment but does not eliminate it.

Specified but Not Built. Several mechanisms described in the Living Taxonomy section (Section 5) are specifications rather than running code. The `Judge#efe0` primitive is formally defined as a Ring-0 evaluator but has no scoring implementation: quality assessment in the current system depends on manual review or external LLM evaluation. The Genesis Loop (Hypothesize $\rightarrow$ Judge $\rightarrow$ Harden $\rightarrow$ Mint) is a protocol for vocabulary self-evolution but is not automated; pattern creation remains human- or agent-initiated through `sema_mint`. The Experience Matrix, which proposes content-addressed tensor transfer between agents ("Experiential Telepathy," Section 5.4), requires serialization infrastructure not yet built. These represent the system's most speculative claims and require experimental validation before deployment.

Built but Not Enforced. 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 simply by not calling the handshake tool. As discussed in Section 3.4, fail-closed is a property of compliant deployments rather than an unconditional property of the protocol. Mandatory enforcement would require integration into the agent's execution harness, not just its tool set—a deployment choice beyond the scope of Sema itself.

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. Although this paper presents Sema as inter-agent coordination infrastructure, the vocabulary's Mind layer may also serve as an internal scaffold for individual agents: hydrated reasoning patterns carry explicit failure modes and invariants into the agent's own context. This use case is not evaluated here; the theoretical foundations for composing reasoning across solvers are developed in [16].

Empirical Pattern Validation. The methodology described in Section 4 validates patterns analytically—by review, rules, guidelines, database/DAG constraints, and broad-use tests—but not empirically. We do not currently measure whether a given pattern, once minted, actually improves downstream agent behavior: whether tasks coordinated through `StateLock#5602` succeed more reliably than those coordinated through ad-hoc prose, whether a Mind-layer reasoning pattern produces better solutions than unstructured chain-of-thought, or whether a given formulation of a mechanism is clearer to agents than its alternatives. This is a clear future-work direction. An empirical layer would treat each pattern as a testable hypothesis, A/B-comparing variants on realistic agent benchmarks and feeding the results back into the next round of mechanism refinement. The refinement pass described in Section 4.8 sharpens patterns against shared formal criteria; an empirical refinement stage would sharpen them against observed task outcomes, producing a population of patterns that are not merely internally coherent but demonstrably useful.

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 20 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 canonical 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#efe0` primitive and adversarial hardening (Section 7.5) 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.

The Synonymy Limit. Sema proves identity, not canonical uniqueness. Two Pattern Cards that describe behaviorally equivalent mechanisms in different words—different field orderings notwithstanding canonicalization, different invariant phrasings, different decompositions into primitives—will produce different hashes and therefore count as different patterns. The protocol guarantees that holders of the same hash reference the same canonical definition, but does not guarantee that holders of behaviorally equivalent meanings hold the same hash. Detecting and merging such synonyms is a curation task carried by the Taxonomist role and the embedding-similarity audits described in Section 4, not by the protocol itself. Refinement reduces synonymy in the bootstrap library, but the protocol remains agnostic to it: a community fork producing patterns equivalent to the canonical library would coexist with rather than collide with that library, and merging the two requires explicit alignment work outside the cryptographic layer.

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#efe0` primitive is designed to address this, but its effectiveness as an automated quality gate remains unvalidated.

Alignment Positioning. A content-addressed vocabulary of *beneficial* strategies—patterns for conflict de-escalation, informed consent, non-coercive persuasion, care under uncertainty, adversarial input handling—would carry the same guarantees the coordination library already provides: hash equality means exactly the same canonical behavior specification, mismatch halts, composition is typed. Building such a library is a straightforward application of the protocol presented here, and we expect it to be a fruitful direction. We are careful, however, not to overclaim Sema's role in alignment. Sema's guarantees are about *semantic preciseness*—two agents provably reference the same canonical definition by a given handle—not about *value alignment*, which concerns whether the meaning they share is benign. The same vocabulary that lets aligned agents coordinate beneficial strategy definitions precisely would let misaligned agents coordinate harmful ones equally precisely. The safety of a Sema-equipped coordination stack is therefore bounded above by the alignment of the models running inside it: Sema is scaffolding for preciseness, and the alignment of the underlying agent is what determines what that preciseness is used for. The metacognitive training proposal in [23]—weaving beneficial values into the model's cognitive architecture through $P(\text{text}, \text{thinking}|\text{context})$ training—remains our primary architectural recommendation for alignment, and Sema is complementary to it rather than a substitute. A vocabulary of beneficial strategies gives an aligned agent vetted patterns to reach for at runtime, but it does not make a

fearful-by-default agent careful. Entangled alignment is the foundation; a Sema library of beneficial strategies is the toolshelf.

10 Conclusion: The Substrate, Not the Law

Sema offers a minimal primitive—content-addressed definition identity—with cascading consequences:

By hashing the definition to get the word, external semantic contracts become verifiable. Because different bytes reveal definitional divergence, mismatch 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 definition—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 definition verification native to the medium of machine thought. Any channel that carries text can carry Sema words; 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 452 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 definitions enables a flourishing, self-governing commons of thought.

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

The vocabulary, interactive graph, and source code are available at `https://semahash.org`.[6]

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-256:560229dacf3ebeebd3461dd3c4a9553d3658bf0f447741b31e38e0418db45e3a

```
{
  "dependencies": {
    "references": {
      "actor": "sema:Actor#mh:SHA-256:57
          f6b9dbc9b8fc596a5238327f29fc24e27d4e0228bfed7352d41f0bfacd4ea4",
      "backoff": "sema:Backoff#mh:SHA-256:
          c6d1d5ba3a4d0347d9191a13c52d9eb43b7b3a19ed611fbb17a0e4dd49249482",
      "cooldown": "sema:Cooldown#mh:SHA-256:6
          eb28c0493afc961191d1b5585d358ee6df332758229c56f75829c0e6b777b52",
```

[6] Source code: `https://github.com/emergent-wisdom/sema`.

```
      "lock": "sema:Lock#mh:SHA-256:051
          cd882b7775ef88bb6dde8864de2749543af6813ec01703e8bc3bccd775de4",
      "state": "sema:State#mh:SHA-256:4
          d582a0ac4af7ae886c83da9825e07c39f1e72ece21fd65a40b6a4fc71882721"
    }
  },
  "signature": [
    "Lock(State)"
  ],
  "mechanism": "A coordination pattern where two {{actor}}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",
  "failure_modes": [
    "Deadlock: both parties wait for each other indefinitely.",
    "Livelock: lock acquired and released rapidly, denying access to others.",
    "Premature dissolution: state auto-dissolves while legitimate work is in
        progress."
  ]
}
```


A.2 sema:FractalIntelligence#mh:SHA-256:c9bc993c4275e181cf597ef9d481c2a9c2a462ea60d70c315e932fe26219dd0f


```
{
  "dependencies": {
    "composes_with": {
      "conceptual_decomposition": "sema:ConceptualDecomposition#mh:SHA-256:75
          bcbfab216222b2630b35dfc2c8b1bc1f24d9c4130913d402db14fa7fc23c3a",
      "localized_learning": "sema:LocalizedLearning#mh:SHA-256:53
          a7f5f86a370c0b6cb14bb1e0196c0e18ae32b2e637b4a870d7b91c1482933b",
      "marginal_value_rule": "sema:MarginalValueRule#mh:SHA-256:311
          b06bbfabf2310065755ebf60ddb02d0ac101870f589c092731a19237fe4aa",
      "polymorphic_solver": "sema:PolymorphicSolver#mh:SHA-256:
          d1048d3e083414f5b7a80f2dad07db242bb03fc8107eca2663d86a7151166b73",
      "problem_framer": "sema:ProblemFramer#mh:SHA-256:5232182564
          bc0765b9ca38a207f1c3680f60b2bfa0628bd546ed558fbca34809",
      "reason": "sema:Reason#mh:SHA-256:
          e9015c8660178539f862e6def248349a654f07571a2559bd8c47c13f9eb17f30",
      "recursion_dive": "sema:RecursionDive#mh:SHA-256:038483584395139
          ff2a1766902d80a65edcab12a1ee39b57614fc93ce704e4ce",
      "reframe": "sema:Reframe#mh:SHA-256:44
          c576a9631661094e0b507c6e2ec3a9cfd5761ab217fa2f32628c9dd4c9db88",
      "state_snapshot": "sema:StateSnapshot#mh:SHA-256:5
          a11e42d46f259b1e618969b879bf1e1143349efee16bfe29af680d25db01d2d",
      "synthesis": "sema:Synthesis#mh:SHA-256:26
          b9e3bc68112d12015efd608c76860f1a5a796c5171145eb2959627fee88a90"
    },
    "references": {
      "agent": "sema:Agent#mh:SHA-256:20723
          f0bbc86b2a6c4e6163dbe70ce1344f06cc1edda1a3f3285b0cf54bff883",
      "conservation": "sema:Conservation#mh:SHA-256:
          ba2edbdbf71a22b97fac45165ec000693c600499cd525ef1a17975a3951d9315",
      "experience_sharding": "sema:ExperienceSharding#mh:SHA-256:1
          b5b67d8c6d94ebc1fa53e4f20c84f6cf1d470ab78dd34d38ef4b4dc3f60cc48",
      "specialize": "sema:Specialize#mh:SHA-256:1
          a76cd0c846c2d7b5325919bad123748366b53ec096e0786a96ab2edbd5fb208",
      "strategy": "sema:Strategy#mh:SHA-256:
          cd1d23e6a0319d00f13415889419d9ebe2ae4706bf58896a8b7008f3a9cbfada",
      "system": "sema:System#mh:SHA-256:
          e314d24e05d0e9ffaaa9c44b249bca8882f00ae6596af18edd245a4fe9df5f0e",
```

```
      "task": "sema:Task#mh:SHA-256:
          b290c7f79fd936d3a3035c5b35658f94b85b6e1b749b1916576a1cf587d8e8c5",
      "universal_solver_tree": "sema:UniversalSolverTree#mh:SHA-256:
          c5233490fe5067f8241c39135bdfb58032df02a30e34a1cebc91155369270cad"
    }
  },
  "signature": [
    "System(Reason)"
  ],
  "mechanism": "Expansion of cognitive capability through {{
      conceptual_decomposition}}: a concept (problem or task) is broken into
      contract-bound sub-concepts, each governed by the same five-surface Solver
      Contract (Manifest, Execute, Consult, Verify, Feedback) that governs the
      parent. A few {{agent}}s can assign solver roles to themselves and perform
      lightweight fractal intelligence for a specific problem \u2014 the resulting
      structure may persist as a reusable pattern that improves through use, or may
       be torn down at completion; both are legitimate modes. The unified {{system
      }} of scalable cognition uses {{reason}} to orchestrate fractal expansion
      within the {{universal_solver_tree}}. A {{problem_framer}} initiates by
      formulating a high-level {{strategy}} before assigning a {{polymorphic_solver
      }} to a {{task}}; the solver executes a {{recursion_dive}} to spawn child
      nodes, each applying {{specialize}} with {{localized_learning}}, while {{
      experience_sharding}} and {{synthesis}} preserve global coherence. {{
      state_snapshot}} provides crash recovery for persistent instances. {{
      marginal_value_rule}} governs recursion depth. On failure, {{reframe}}
      restructures the tree.",
  "gloss": "Expansion of cognitive capability through recursive decomposition of
      concepts into contract-bounded sub-concepts",
  "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."
  ]
}
```

A.3 sema:SpectralTune#mh:SHA-256:b25afcef8b09a2a4368090150d529379ce3124ae280a660ba40b27025265a34f

```
{
  "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": "PositiveInteger",
      "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-256:2d13b593c6209c12f0139e72867ec1acab3bda91b54584ce2b4c6de4f6aea27a

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

```
{
  "dependencies": {
    "references": {
      "agent": "sema:Agent#mh:SHA-256:20723
          f0bbc86b2a6c4e6163dbe70ce1344f06cc1edda1a3f3285b0cf54bff883",
      "belief": "sema:Belief#mh:SHA-256:1
          f44935a0c3f58adb07024c4c3c8df389f2b04d34e29218f6bbb454bbb8e8bfd",
      "check": "sema:Check#mh:SHA-256:5
          f38a1da12b7230618159ba1ff08a4ded1e65b0c6a26af63a4c0d12c11e51d8e",
      "cognitive_bias": "sema:CognitiveBias#mh:SHA-256:4
          b32f79732d4468fa0de4facb106dbbd2889bfa1d3ce62e77b09e606011680c9",
      "compatibility_check": "sema:CompatibilityCheck#mh:SHA-256:3
          abb595523dc765c8ebe497169cd2c12494c93a85e50c8785577cef22c465554",
      "critique": "sema:Critique#mh:SHA-256:4
          e43011af49e8973c5f934ad6ee91ea40254fa1a4f6bff9f6880309ba9affe01",
      "decision": "sema:Decision#mh:SHA-256:934
          e336cd9323a59a2af8f1248d0a4122183efe479c561162289f23d4226944c",
      "loop": "sema:Loop#mh:SHA-256:
          d8144765ffd478df48287cea73974e30b490a039c6eeb4155d56adb2108da27b",
      "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": "Post-decision adversarial check: revise if counter-argument exceeds
     validity threshold",
  "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-256:89358b62fc3e7f54c787f1bfac45970017261d994869adc77c994998e91b75c1

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

```
{
  "dependencies": {
    "composes_with": {
      "atomic_bid": "sema:AtomicBid#mh:SHA-256:
         c429c0ee44df1f2b464558747430e559e80b9f261ebdc88520486eec13307971",
      "compensate": "sema:Compensate#mh:SHA-256:
         b4c51f5f3f9fbc1bfbcc1818e9adad4aef185bff315eaacee47a4a8fbee50631",
      "compute_budget": "sema:ComputeBudget#mh:SHA-256:
         ff07d8bf174ca020ec8e1e2cd9a7ea7575264d531256aba7a1dbc11eecec1f98",
      "pathway_memory": "sema:PathwayMemory#mh:SHA-256:7899805
         e7e4497c1bbd072db6763026b3f0b966e773b1901bdf40701f21861d9",
      "reflexion": "sema:Reflexion#mh:SHA-256:1458
         ddf10e51e9a5dbc4f63d328b8830c41fb24892ca9bef22179710acb4ebeb"
    },
    "references": {
      "parallel": "sema:Parallel#mh:SHA-256:318115
         c507a765d08bb292f74162bb6427f0756b031b2f516b089f6dbf61a753",
```

```
      "polymorphic_solver": "sema:PolymorphicSolver#mh:SHA-256:
          d1048d3e083414f5b7a80f2dad07db242bb03fc8107eca2663d86a7151166b73",
      "rigorous_solver": "sema:RigorousSolver#mh:SHA-256:
          b7aa2d189946a424556643b92a344b3ad4fd501f41ebeb9961a4162118ed35f3"
    }
  },
  "mechanism": "A high-velocity implementation of {{polymorphic_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. A {{pathway_memory}}
      accumulates across runs so the optimistic route-selection converges on
      strategies that historically succeeded under similar conditions.",
  "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."
  ],
  "derived_from": "sema:Solver#mh:SHA-256:
      b00a16afef5a9d3293955f36b44fe6fc0e0c4ce0fc118de4a7459f77fe7d98d7"
}
```

B Category Taxonomy

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

The 452 patterns are organized into 12 category names across 13 layer-category paths and 4 fundamental layers:

Physics Layer (17 patterns) The immutable substrate.

- **Primitives** (16): Attractor, Causation, Conservation, Dampen, Decay, Distance, Entropy, Equilibrium, Gradient, Lock, Measurement, Mutex, MutualInformation, Noise, PhaseTransition, Reversibility.
- **Time** (1): CausalBarrier.

Infrastructure Layer (151 patterns) Operational constraints.

- **Data Structures** (93): AcceptSpec, Aesthetics, Anomaly, Artifact, Assessment, Assumption, Audit, Ballot, Belief, Boolean, Break, Cache, Card, Category, Chain, CognitiveBias, ConceptAnchor, Condition, Constraint, Context, Contract, Correlation, Criteria, Cyclic, DAG, Datum, Decision, Event, Exception, ExecutionManifest, FailureTrace, Forest, FrameSpec, Goal, Hierarchy, Hypothesis, Identity, Ledger, MECE, MechanisticDesignProposal, Message, Meta, Metric, Mode, Nature, Option, Outcome, Overlap, Parallel, PerformanceSignal, Permission, Plan, Probability, Problem, ProblemSpace, Prompt, Proposal, ProtoPack, Protocol, Prototype, Queue, Resource, Result, Risk, RolloutManifest, RuleSet, Score, ScoringFunction, Sequence, Shard, Signal, Skeleton, Snapshot, Solution, SolverManifest, Spec, State, Status, Step, Stream, StyleSpec, Subject, Summary, System, Task, Tension, Topology, Transition, Tree, Value, Variable, Vector, Work.
- **Primitives** (49): Act, Actor, Aggregate, Backoff, Branch, Budget, Care, Check, CircuitBreaker, Combine, Compare, Compensate, Compress, Cooldown, EntropyPump, FailClosed, Feedback, FeedbackSignal, Gate,

Greet, Heartbeat, Hysteresis, IdempotentWrite, Incongruity, Judge, Loop, Monitor, MonitorReport, NegativeProof, Observe, Probe, Quorum, Rank, ReAttempt, Route, Sandbox, Search, Select, Sign, StateAudit, StateSnapshot, StateTransition, TaskLifecycle, Throttle, TimeWarpLog, ToolInvoke, Trace, TriGate, Warmup.

- **Verification** (9): AuditTrail, CompatibilityCheck, ExplainBeacon, HumanApprove, InputGuard, OathBind, OutputGuard, SpotAudit, Validate.

Mind Layer (178 patterns) Hybrid cognition—always decomposable and delegatable.

- **Inference** (22): BaseRateInclude, BayesUpdate, BreadthGovernor, ConfidenceCalibrate, ConfirmationBlock, ContextFirst, EpistemicCalibrate, HackDetect, HindsightBlock, LayeredCheck, NormCheck, NormativeJudge, OntologyAdapt, ProphetFanOut, RegimeSense, ScopeFreeze, SemanticTabu, SourceEvaluate, SurprisalUpdate, SurvivorCorrect, TemporalEnsembleForecasting, TruthseekingProtocol.
- **Memory** (15): BeliefTracking, ChunkMerge, ContextCompress, CurriculumReplay, ExperienceSharding, HolographicShard, LatentAttachment, LocalizedLearning, PathwayMemory, Proprioception, RetrievalAugment, Scratchpad, SelfReminder, SimulationTrace, TraceBelief.
- **Reasoning** (60): Abduction, BackwardChain, Bisect, ChainOfThought, CiteBack, CognitiveEcho, CollaborativeWritingProtocol, ConceptualDecomposition, ConstructOntology, Critique, Decompose, DecompositionGate, Deduction, DeepResearch, Dialectic, Eliminate, Estimate, EthicalReasoningProtocol, Expansive, ExtendedThinking, Fermi, FirstPrinciples, FrameError, Generalize, GraphOfThought, HeuristicSnap, HumanEmulatorProtocol, Induction, Interpret, Invert, LeastToMost, LivedProof, MetaPrompt, Parsimony, PatternDiscovery, ReAct, Realizable, Reason, RecursionDive, RecursiveRootCause, Refine, Reflexion, Reframe, RequestFraming, SelfConsistency, SkeletonOfThought, SocraticLoop, Specialize, SteelmanCheck, StepBack, StrategicReading, Summarize, Synthesis, Think, Translate, TreeOfThoughts, Uncertain, Understand, Verification, WhyClimb.
- **Strategy** (81): AdversarialSteel, Agent, AnalogyBridge, AntifragileInversion, BeamSearch, Bubble, Build, CapacityPressure, CommitmentDevice, Compose, ComputeBudget, ConceptBlend, ConstraintFirst, ContingencyPlan, Creative, CreativeBlend, Crystallize, Deep, Defer, DepthGovernor, DesignArchitect, DiscoveryProtocol, DogfoodFirst, EmpathySim, EmpiricalTest, EpistemicROI, EventReact, Experiment, ExploreExploit, Falsification, FractalIntelligence, HypothesisEngine, HypothesisLadder, Jester, Kairos, LatentWander, LateralOptimization, ManifestPlanning, MarginalValueRule, MentalSim, MetaCheck, MetaProtocols, NoiseInjection, Novelty, OODA, OpportunityCost, OptimalStop, Optimize, PURE, PUREBrainstorming, PURECheck, PUREOptimization, Parallelize, ParetoFront, PerspectiveEnsemble, PolymorphicSolver, PreMortem, Prioritize, ProblemFramer, RedTeam, Reflex, RegretMinimization, RepresentationSwap, Retry, RigorousSolver, Roadmap, RootSolver, SacrificialProbe, Satisfice, Silence, Simulation, Solver, SteelmanFirst, Strategy, SunkCostIgnore, TensionHold, ThinSlice, TimeboxThink, TradeOff, UncertaintyMap, WorldReversible.

Society Layer (106 patterns) Multi-agent coordination.

- **Coordination** (12): Compromise, Consensus, ConsensusFinder, Delegate, Disband, Elect, IdentityHandshake, LazyConsensus, OntologyHandshake, Rally, Resonate, Vote.
- **Economics** (10): AtomicBid, AttentionMarkets, Award, Bid, ContinuousResourceAuction, ExchangeRate, Gardener, MintWhenFriction, ValuePeg, Yield.
- **Governance** (8): AnchorDrop, Constitution, DocumentedOverride, Responsibility, Role, SolverTree, UniversalSolverTree, WorldTransparent.
- **Protocols** (76): AdversarialProof, AgentDiscover, AgentProtocol, AgentSandbox, AmbiguityResolution, Axiom, BearerToken, BoundedTask, Canary, ConfusedDeputy, ContextSwitch, CounterfactualAnchor, DataMinimization, DeliberativeAlign, Deploy, Discover, DissentSeek, DriftWatch, EbbFlowSync, EjectionSeat, EvaluatorOptimizer, ExpiringToken, FabricSharding, FeatureFlag, GenealogicalTrace, GlacialVault, Global, GracefulDegradation, Handoff, HeldRelease, IntentGap, InternalConsistency, InvariantFilter, LatticeCommit, MemeticSeed, ModestClaim, MonotonicCounter, Nucleate, OptimisticSolver, Oracle, OrchestrationLoop, OsmoticFilter, PatternEmergence, PatternSketch, PermissionEscalate, PhasedRefinement, PromiseGraph, PromptChain, PropheticQuorum, QuorumPulse, RealizationProtocol, ReceptivityGate, ReversibilityCheck, Robustness, Rollout, RolloutWatch, RootHashGossip, ShoutWhisper, SignalReflection, SolverNode, SomaticMarker, SpectralTune, StateLock, Stigmergy, StructuralCoaching, SynergisticMode, Taper, ThreeLevelCollision, TieredAccess, ToolDiscovery, TranslationProxy, UniqueHandle, UptakeAsGround, UptakeOverTimestamp, WorkerMode, Workflow.

References

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

[2] Mingqian Feng, Xiaodong Liu, Weiwei Yang, Jialin Song, Xuekai Zhu, Chenliang Xu, and Jianfeng Gao. SEMA: Simple yet effective learning for multi-turn jailbreak attacks. In *International Conference on Learning Representations (ICLR)*, 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. Originating statement of the robustness principle: "be conservative in what you do, be liberal in what you accept from others." The shorter "be conservative in what you send, be liberal in what you accept" paraphrase is from RFC 1122 (1989).

[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. Byzantine consensus study: LLM agents fail to reliably reach valid agreement even in benign no-stake settings; performance degrades with group size and collapses under a single Byzantine agent.

[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] 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.

[34] 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.

[35] 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.

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

[37] 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.

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

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

[40] 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.

[41] 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.

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

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

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

[45] 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.

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

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

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

[49] 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.

[50] 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.

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

[52] 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.

[53] Zhenhua Zou, Zhuotao Liu, Lepeng Zhao, and Qiuyang Zhan. BlockA2A: Towards secure and verifiable agent-to-agent interoperability, 2025. Blockchain-anchored trust framework using DIDs and smart contracts to verify message authenticity and execution integrity in A2A interoperability; hashes interaction artifacts rather than definitions.

[54] Amjad Fatmi. Faramesh: A protocol-agnostic execution control plane for autonomous agent systems, 2026. Introduces the Action Authorization Boundary (AAB): a non-bypassable, deterministic PERMIT/DEFER/DENY decision over a Canonical Action Representation, placed between agent reasoning and real-world execution.

[55] Om Tailor. Verifier-bound communication for LLM agents: Certified bounds on covert signaling, 2026. Protocol that gates messages (not agents) into the shared transcript via a verifier-approved predicate, producing certified upper bounds on covert signaling between colluding LLM agents.

[56] Anthropic. Agent skills specification. `https://agentskills.io/specification`, 2025. Open standard released December 2025 for portable capability definitions via `SKILL.md` files (YAML frontmatter + Markdown body) with progressive disclosure; adopted by OpenAI, Microsoft, Cursor, and others.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.