SEMA: WHEN THE HASH IS THE WORD

A PREPRINT

Henrik Westerberg
Emergent Wisdom
henrik.westerberg@emergentwisdom.org

April 7, 2026
Revised: November 17, 2026

ABSTRACT

Autonomous agents often reason and coordinate through labels that do not reveal which external information they denote. We present Sema, a content-addressed reference system for reasoning and communication that creates *hash-based verifiable words*: participants encode and hash information under an agreed representation, then reuse the content address as a term, optionally with a human-readable handle. Sema can use any representation if participants agree on how content is encoded, hashed, resolved, and decoded. This paper develops one concrete realization based on canonicalized JSON-compatible objects, field-level Merkle commitments, explicit identity rules, and content-addressed references. Full-reference verification establishes identity of the resolved hashed content, while semantic equivalence, correctness, and compliance remain separate questions.

As a worked application of the Sema primitive, we introduce Pattern Cards, a concrete schema for deliberate, System 2-style reasoning. They make often-skipped reasoning moves explicit through contracts for mechanisms, conditions, invariants, typed dependencies, and failure modes. Within one agent, resolving a card can make a reusable reasoning procedure explicit. Across agents, the same content-addressed definitions support coordination: agents can verify that they resolved the same definitions; in strict mode, an enforced handshake halts on mismatch.

A bootstrap library of more than six hundred cards demonstrates the design. Its contract fields averaged 26.7 times the tokens of short wire references—a transmission-compression result, not an inference-token saving, because definitions were hydrated before use. Illustrative agent demonstrations examine coordination behavior. The same primitive could support a Living Taxonomy of Thought; its utility, evolution, and governance remain open questions.

Keywords Hash-Based Verifiable Words, Content-Addressed Semantics, Multi-Agent Systems, Verifiable References, Semantic Coordination, Merkle Commitments, Agent Vocabularies, Pattern Cards

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 an ordinary label in an agent message does not reveal which external definition its sender intends. Without an explicit resolution and verification step, apparently shared terms can conceal different specifications. Five problems motivate such a step:

The Paraphrase Problem. Agent A may need to reproduce a lengthy behavioral specification for Agent B, and each paraphrase risks drifting from the last. A reusable reference could carry the identity of that specification more compactly.

The Identity Problem. When two agents use the phrase "coordinate safely," the label alone does not establish that they refer to the same external definition. Silent divergence can compound through agent chains where identical labels hide different specifications.

The Hallucination Problem. Large Language Models fabricate references and citations [1], motivating a "fail-closed" architecture that checks references against their resolved content. A fabricated or mismatching pointer can then be rejected.

The Trust Problem. RDF and OWL provide shared representation frameworks, but do not by themselves verify that low-trust peers resolved the same external definition.

The Granularity Problem. A generic report that "the coordination failed" leaves the cause open, whereas "the Lock Invariant in `StateLock#c9c2` was violated at $t = 3.2$s" identifies the clause being assessed. A vocabulary of named contracts gives agents and their evaluators a more precise diagnostic language.

Sema[1] is a communication primitive: hash information under an agreed representation and carry the resulting content address directly in communication. A term made this way is a *hash-based verifiable word*, or a verifiable word for short: a word whose identity is the hash of its definition, normally written as a readable name joined to that hash. What can be verified is that two parties hold the same definition, not that they read it alike or that it is correct. To *resolve* a verifiable word is to fetch the definition its hash points to; to verify it is to recompute the hash from what was fetched. This paper develops a JSON/Merkle realization for behavioral definitions called Pattern Cards.

An agent writes: "I `Delegate#00fa` this `Task#a850` to you, expecting a `Solution#8d55` that satisfies this `AcceptSpec#37fa`." Each anchored term is both a readable word and a verifiable reference to a Pattern Card. The four characters after the `#` are the first characters of the SHA-256 hash of the card's definition.

Content-addressing definitions has clear precedents in Hawke's Secure Definition Hash (SDH) [3], Trusty URIs [4], and systems such as Git, IPFS [5], and Unison [6]. Sema builds on this machinery to make readable, hash-anchored terms part of the natural-language stream agents write in. A minted term, one made by writing a definition and hashing it, can name a reusable procedure together with its conditions of use—invariants, preconditions, and failure modes. The contribution developed here combines those lexical behavioral contracts with field-level comparison, exact dependency commitments, and a handshake that an execution harness can enforce.

In the JSON/Merkle realization, a change to a hashed canonical field produces a different digest with overwhelming probability. Comparing resolved definitions can therefore expose mismatch before coordination depends on them (Figure 1).[2] The general reference form is:

$$\text{full-reference}_\sigma(\ell, x) = \ell \parallel \sigma \parallel \mathcal{H}_\sigma(\text{Encode}_\sigma(x)) \tag{1}$$

Here $\parallel$ denotes concatenation, ℓ is an optional readable label, and σ names the agreed encoding and hash scheme. Prose may abbreviate the digest as a short wire reference, but collision-resistant verification resolves that abbreviation to the full reference or an agreed full root.

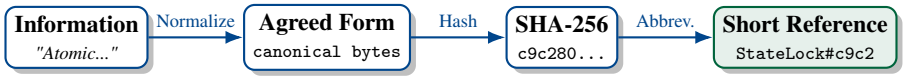

Figure 1: The content-addressing pipeline used in this paper. The digest is derived from an agreed representation and paired with a human-readable handle; the four-character form is a wire reference, while full-digest or full-root verification remains the security boundary.

The practical consequence is that a vocabulary of Pattern Cards behaves like an agent framework mixed into English. Agents can use as much structure as the moment requires, naming recurring mechanisms and leaving contextual prose freeform. When a recurring situation demands a different contract, an agent can mint a linked variant rather than paraphrase or overload the existing term. Once the relevant definitions are available, a few anchored words can replace repeated explanations of those definitions; Section 7.4 measures this wire-reference economy across five contract fields. It does not measure the preservation of every contextual detail when a whole passage is rewritten.

Ordinary communication leaves many reusable procedures unnamed or defined anew at each use. A resolvable agent vocabulary makes additional names practical: detailed definitions can be expanded on demand, allowing recurring

[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 and operate at different layers (definition-identity verification vs. attack methodology).

[2]A large context window can carry definitions, but capacity alone does not perform a cross-agent identity check. A full-reference comparison makes that check explicit. Pattern references may also serve as attention anchors; whether that improves reasoning is a separate empirical question.

procedures to become precise, composable terms without requiring every participant to memorize the whole library. It also makes room for coined words: many concepts an agent needs often are needed too rarely in everyday speech to have earned a word. `Brakefade#764a`, for example, names an activity that continues while the practical ability to slow, stop, or reverse it diminishes. The handle alone does not specify its exact definition: the card's gloss states the concept in plain English, so a reader new to the word resolves it rather than guesses.

The same substrate can act as collective memory across summarization. Prose may shift at each compaction boundary, while a preserved full identifier remains pinned to the same verified definition for as long as its content remains resolvable. Sessions and agents can therefore retain stable conceptual anchors rather than relying only on surviving phrasing.

Pattern Cards offer three proposed channels of reasoning benefit. First, verified reference identity lets a conclusion cite the exact external definitions it used and exposes mismatch before work proceeds from a false shared assumption. Second, explicit invariants provide a mid-task checklist against which an agent or evaluator can test the work. Third, named procedures make often-implicit reasoning moves easier to request, combine, and revisit. These benefits depend on use: re-supplying an already clear definition may add context tokens without improving the result.

Models already produce fluent reasoning chains, and their fluency can allow unchecked premises to pass review. Pattern Cards are intended to make the steps explicit and enforceable by a surrounding workflow: citable by hash, checkable against a contract, and usable as inputs to a gate. The intended benefit lies in multi-step work, coordination, and long-horizon tasks rather than single-pass scoring over well-known material. Some of that benefit falls outside answer-only metrics: a verified refusal prevents an action or assertion rather than producing a scored answer.

The corrective-procedure intuition is familiar from human reasoning: a named move such as checking base rates, looking for survivorship bias, or seeking a counter-argument gives a reasoner a specific action to take. Pattern Cards extend this checklist idea into a resolvable vocabulary whose procedures are named, composable, and cumulatively revised. Their intended contribution to language-model reasoning is reliable access to such moves, not a claim that a model cannot infer an omitted step.

Sema's agent-coordination application assumes a low-trust environment where agents may hallucinate, drift, or actively deceive. This is why the bootstrap library is closer in intent to a "Constitution for a Digital City" than a passive "Library Catalog." Patterns such as `LatticeCommit#488a` and `ReceptivityGate#0db5` specify how participants should handle integrity and admission, while reasoning cards specify moves an individual agent can invoke. The metaphor concerns explicit rules available for use and inspection; their authority, adoption, and enforcement remain decisions of the participating community.

This use is the first operational slice of a larger architecture: a Living Taxonomy of Thought. A Sema-equipped model could represent a book, theory, or strategy as a graph of referenced concepts, mechanisms, assumptions, and disagreements. Where an existing pattern fits exactly, the model reuses it; where the fit is only partial and the difference changes the concept's contract, it mints a linked variant rather than overloading the old word. Successive agents can therefore inherit not only documents, but an inspectable record of the conceptual distinctions extracted from them. Sema supplements rather than replaces the source material, and the resulting taxonomy need not be final: source-specific patterns may remain local, competing interpretations may coexist, and only broadly reusable concepts need enter a shared library. This paper implements the identity, resolution, composition, and minting substrate for that architecture; training models to construct and reason through work-scale pattern graphs remains future work.

Section 2 states the problem and five desiderata. Section 3 gives the protocol: what a Pattern Card is, how it is hashed, how one is found, and the fail-closed handshake. Section 4 presents the bootstrap library, and Section 5 the Living Taxonomy it is meant to grow into, an idea first sketched in earlier work [7]. Section 6 describes the implementation and says what is built and what is only proposed; Section 7 gives the evidence, layer by layer. The paper's seven contributions are listed after the related work, in Section 8.6, and Section 9 gives the limitations.

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. In the definition-coordination application developed here, full-reference verification instead establishes that agents resolved the same hashed external definition, 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 rather than proceeding with different interpretations. *Compactness* (D3): for nontrivial definitions, wire references should be smaller than the content 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. These answer the problems of the Introduction as follows. The Paraphrase Problem asks for Compactness; the Identity and Trust Problems ask for Identity and Fail-Closed Coordination; the Hallucination Problem asks for Fail-Closed Coordination and Resolvability. The Granularity Problem is answered by what is hashed, a contract with named clauses (Section 3.2), and Permissionless Minting answers the need for words that do not yet exist.

3 The Sema Protocol

This section develops the Pattern Card (Section 3.2), its representation and hashing rules (Section 3.3), discovery (Section 3.4), the fail-closed handshake (Section 3.5), and the type system used by the bootstrap library (Section 3.6).

3.1 Design Principles

Sema's core rule is representation-agnostic: hash information under an agreed representation and scheme, then use the resulting content address directly in communication. It is not restricted to JSON or to a fixed schema. Raw bytes, a canonical graph, or a structured object can each participate if recipients know how to resolve the reference and recompute its identity. The implementation used in this paper supplies deterministic recursive Merkle hashing for JSON-compatible objects; a molecular database, legal contract repository, or game-rule engine could define a different representation while retaining the same communication primitive. Sema is closer to Unicode than to a universal language: it standardizes a way to carry identity, not every vocabulary spoken through it.

At the wire level, the hash functions as a word in the natural-language stream that agents exchange; the LLM does not reason directly over hash bytes. To use the referenced procedure, an agent or host runtime resolves the definition and places the relevant text in model context. The deployment pattern assumed here is:

$$\text{Prompt} = \text{System} \oplus \text{Hydrate}(\text{Patterns}) \oplus \text{Query} \tag{3}$$

Here $\oplus$ joins the parts of the prompt in order. The *wire-level token*, such as `PropheticQuorum#48df`, stays compact in communication; its resolved definition occupies model context. The design thus trades context-window tokens for wire-level integrity, with assurance supplied by full-identifier or agreed-root verification.

The communication primitive, the JSON/Merkle realization, the Pattern Card schema, and the *Sema Bootstrap Library* are distinct layers. Pattern Cards are this paper's concrete agent-coordination schema; the library in Section 4 is a bootstrapped opinion on useful definitions expressed through that schema. A *pattern* is a reusable definition, and its Pattern Card is the record that holds it; the paper says "pattern" or "card" as the context suits. Coordination requires a shared Schelling point, so we provide 617 patterns designed to bootstrap agent society. Their fields describe actionable behavioral contracts rather than passive entries: the `mechanism` describes how the pattern works, the `invariants` state what must remain true, and the `dependencies` state how a pattern composes. Execution and enforcement still require an agent, evaluator, implementation binding, or surrounding harness. What follows describes the JSON/Merkle mechanics and this concrete instantiation.

3.2 Pattern Cards

A Sema Pattern is not merely a dictionary definition; it is a structured behavioral specification (Figure 2). Preconditions, postconditions, and invariants make contract clauses explicit for inspection by agents, external evaluators, or executable bindings; the Pattern Card does not itself execute or enforce them. This structure builds on a lineage of structured LLM frameworks—notably DSPy Signatures [8], DSPy Assertions [9], and the Prompt Pattern Catalog [10]—which introduced contract-like constraints for language model pipelines. Sema's contribution is to make canonical contract identity hash-verifiable: including the clauses in the Merkle root permits cached digest comparison without asking a model whether two contract texts mean the same thing. We selected these fields to transform vocabulary from *Passive Knowledge* into explicit contract structure, as illustrated by a simplified version of `PropheticQuorum#48df`

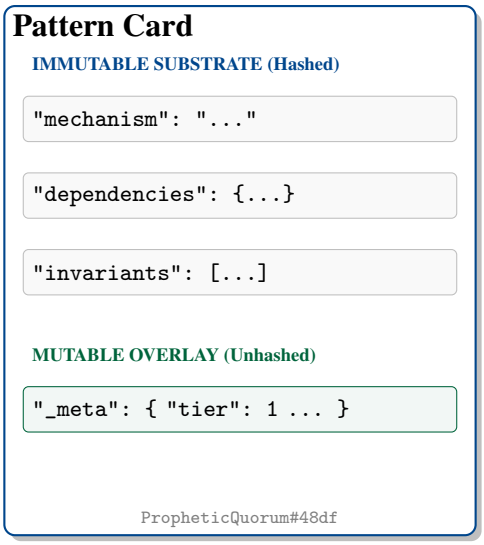

Figure 2: Pattern Card Anatomy. The *Substrate* defines identity (hashed); the *Overlay* enables evolution (unhashed).

whose wiring shows the typed dependency roles; the released card states the idea as a two-phase protocol and cites its dependencies as references:

Dependencies (The Imports) **Rationale:** A categorized map of hard links. Makes declared dependencies machine-readable without parsing natural language.
Example:

```
"dependencies": {
  "composes_with": {
    "sim": "sema:Simulation#mh:SHA-256:1058
        e7090079e846251ed480d7afec0328809d22bebb6b205425b2de614787c3",
    "vote": "sema:Vote#mh:SHA-256:6
        ba4208c942c637524fe4461b98d12fcdaae97eac84b07ddaeff170a16761b53"
  },
  "accepts": {
    "proposal": "sema:Proposal#mh:SHA-256:5
        d0a98055eef6c963d4c9ec0403cac243720ce8927398032c19b4f5d8cc94506"
  },
  "yields": {
    "decision": "sema:Decision#mh:SHA-256:
        b1407127acd4a39314fc24b418310e4b95e0347e4e22f376151ab765e2dfd2c7"
  }
}
```

Mechanism (The Procedure) **Rationale:** Describes how the pattern works, using the keys defined in Dependencies; for this card, an intended procedure.
Example: "Instantiates `{{sim}}` to predict outcomes of `{{proposal}}` before calling `{{vote}}` to produce `{{decision}}`."

Invariants (The Safety Contract) **Rationale:** States conditions that must hold throughout a valid application; checking them requires an agent, evaluator, or executable binding.
Example: "Prediction Precedence: Vote must occur after Simulation."

Preconditions (Entry Conditions) **Rationale:** Records conditions that must hold before application, supporting chaining and planning when an agent or external evaluator checks them.
Illustrative example: "The input state and simulation configuration have been fixed before invocation." A precondition such as "Determinism level is high" additionally needs an agreed criterion for "high."

Gloss (The Embedding Anchor) **Rationale:** States the concept in one plain line, which also serves as the semantic hook for vector retrieval.
Example: "Aligning predictions before aligning votes."

Specialization (The Type Relation) **Rationale:** Records that this definition is a narrower kind of an exact parent definition and emits an `IS_A` edge.
Example: `"extends": "sema:Task#mh:SHA-256:..."`. Because the parent version is named by its full identifier, retargeting the child is an explicit semantic change.

Metadata (The Overlay) **Rationale:** Stores the evolving knowledge graph (associations, suggestions, safety tiers) without affecting the identity hash. A `tier` rates the rigor of a card's contract (Section 4.3); a `ring` marks how settled it is, from ring 0, the most stable, to ring 2, the most fluid.
Example: `"_meta": { "tier": 1, "related": ["Mutex#b031"], "supersedes": ["sema:OldHandle#mh:SHA-256:..."] }`. The unhashed `supersedes` list records version history without making the route by which a definition was authored part of its identity.

Crucially, the Pattern Card schema 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 keeps taxonomy from becoming part of a pattern's permanent identity. The intuition is that the substrate is mathematics and the overlay is politics: equality of the hashed specification of what a pattern does is checked mathematically, while where it belongs remains mutable metadata subject to community judgment. 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#8656` 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`	Required state after a valid invocation
`parameters`	Configurable parameter objects
`failure_modes`	Known ways the pattern can fail
`data_schema`	JSON Schema for runtime data structure
`extends`	Exact parent definition this specializes

3.3 Cryptographic Foundation

Applied to definitions, the communication primitive permits a *Semantic Commons*: a shared namespace where agents actively mint and reference new concepts without requiring a central authority to decide what words exist. Just as Git allows any developer to commit code, Sema allows any participant to mint a content-addressed item under an agreed representation. In this implementation, an agent defines a Pattern Card as a JSON object and computes its Merkle Root from the card's semantic fields, transforming the namespace from a read-only reference into a writable *thoughtspace*.

$$a_\sigma(x) = \text{Hash}_\sigma(\text{Encode}_\sigma(x)) \tag{4}$$

Here σ identifies the agreed representation and hash scheme. The JSON/Merkle implementation specializes this rule to field-addressable Pattern Cards:

$$H_{\text{pattern}} = \text{Merkle}(\{\text{mechanism}, \text{gloss}, \text{dependencies}, \dots\}) \tag{5}$$

This hash becomes the permanent identity component for that object (e.g., `#a1b2`); the readable handle remains separate. If Agent A and Agent B resolve the same full identifier, recompute the hash over the returned object, and obtain a match, they establish—subject to SHA-256 collision resistance—that they hold the same canonical definition, regardless of who created or stored it.

The JSON realization implements content-addressing via a Merkle Tree. Given a definition d and a separate readable label ℓ, the Pattern Card identifier is:

$$\text{id}(\ell, d) = \texttt{sema:}\|\ell\|\texttt{\#mh:SHA-256:}\|\text{RootHash}(d) \tag{6}$$

where ℓ is not part of RootHash(d), and RootHash(d) is computed via recursive Merkle Tree construction using deterministic JSON canonicalization inspired by RFC 8785 (JSON Canonicalization Scheme) [11], adapted with explicit NFC string normalization for deterministic string equivalence, and aligning with principles from RDF Dataset Canonicalization [12] and content-addressed semantic data [13]. This ensures that canonical-equivalent JSON objects produce deterministic hashes regardless of serialization order.

Every Merkle node is hashed with a tag for its type, so that structurally distinct JSON values—for example the string `"1"` and the number `1`, or a two-element list and a one-entry dictionary—cannot share a hash input. Appendix C gives the exact rules of canonicalization v3, with the normative specification and the interoperability vectors.

In the Pattern Card realization, dependency and specialization references, when present, carry their targets' full Sema IDs inside the hashed object. A card carrying such references therefore commits recursively to its exact referenced-definition closure, as cards in the bootstrap library do; the current implementation's aggregate roots below commit, independently of insertion history, to the vocabulary-wide definition set and handle bindings, but not to the mutable `_meta` taxonomy. Neither referenced composition nor vocabulary-wide aggregate roots are requirements of the representation-agnostic Sema primitive.

Vocabulary snapshots require a second canonicalization layer because a set of disconnected pattern DAGs has no traversal-defined order. Beginning with semahash 0.4.0, the implementation therefore publishes two versioned commitments. The *semantic-set root* commits to the sorted set of unique raw pattern digests; the *catalog root* commits to the sorted mapping from each handle to its raw pattern digest. Both are Merkle Tree Hashes in the construction of RFC 9162 [14]; Appendix C gives the tree construction and ordering, and points to the normative specification for each scheme's leaf framing. The scheme identifier and leaf count accompany each root; a bare digest specifies neither its hash function nor its tree construction.

The per-pattern Merkle structure assigns every field a deterministic, collision-resistant hash, allowing agreement and disagreement to be localized even when full pattern roots differ. As a concrete example, suppose Agent A and Agent B both reference the handle `StateLock` but their pattern roots disagree. Field-level hash comparison may reveal matching `invariants` hashes (both definitions state that the lock is exclusive and auto-releases on timeout) while their `failure_modes` hashes diverge (A's pattern enumerates network partition as recoverable; B's treats it as terminal). This identifies precisely which clauses match and which require reconciliation. Selective coordination on a matching subset requires an explicit policy; the current handshake remains fail-closed on root mismatch. Whole-document hashes (SDH [3], Agora [15], BlockA2A [16]) collapse the comparison to a single binary mismatch, whereas the Pattern Card Merkle structure preserves field-level diagnostic granularity.

Fields not hashed include a pattern's own `handle`, `_meta`, and any `sema_*` fields. Renaming a pattern does not change that pattern's own digest. Canonicalization v3 does, however, retain target handles inside hashed structured references: dependency maps contain target handles, and `extends` stores a full Sema ID. Renaming a referenced pattern can therefore change the hashes of both dependents and specialized descendants; graph-wide rename invariance requires a separate versioned canonicalization migration covering both forms.

To keep mechanism text independently comparable as dependencies evolve, the implementation uses a template hashing strategy where the mechanism uses local placeholders like `{{hypothesis}}` representing the algorithm, while the wiring maps these placeholders to specific hashes in the `dependencies` object. The Pattern Identity is the Merkle Root of both, allowing tools to distinguish a logic-field match from a dependency-wiring mismatch when agents use different library versions. Rewiring can change the procedure's behavior even when its mechanism text is unchanged. The identity therefore commits to the wiring itself: which definition each placeholder is bound to. A pairing whose placeholder names its own target, such as `{{vote}}` bound to `Vote`, is implied by the target's entry, provided no other placeholder name in the card folds to the same name; every other pairing, such as `{{sim}}` bound to `Simulation`, is hashed explicitly (Appendix C). Canonicalization v2, used through release 0.10.0, hashed which definitions each dependency category binds, and how many times, but not which placeholder each one fills, so two cards that differed only in that could share an identity. Moving the bootstrap library to v3 changed 55 of its 617 identities: the 8 cards whose placeholders carry role names and the 47 that depend on them. The other 562 are the same under both.

3.4 Discovery

The Pattern Card tooling reconciles two opposing needs: discovery, which benefits from ambiguity and high recall, and coordination, which requires exact identity for the references being verified. It does so through *progressive ambiguity collapse*, captured by the `Taper#5a75` pattern: discovery begins broadly, then the workflow narrows to selected definitions whose identities can be checked exactly.

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, optionally confined to one layer or category, that merges keyword matching with embedding similarity and keeps each result's higher score. This strategy mirrors Blended RAG [17], merging keyword precision with semantic recall [18]. 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 [19]. This hybrid strategy allows approximate intent (e.g., "how to handle errors") to find precise patterns (e.g., `Retry#4f9f`), but necessitates the subsequent verification step.

Finally, in the *Verify* phase, the agent invokes `sema_handshake()` (Section 3.5), which distinguishes cooperative drift detection from strict identity verification. The pipeline uses semantic similarity for discovery and hash-based identity comparison for coordination.

3.5 The Fail-Closed Protocol

When agents coordinate using Sema patterns, the handshake compares the selected external definitions required for the interaction (the "Context"). The initiating agent lists the required pattern identifiers; each agent independently resolves them and deterministically constructs the catalog Merkle Tree of that subset. Each leaf binds a requested handle to the hash of the definition it resolved, so swapping two handles' definitions cannot preserve the root. The agents exchange the declared root scheme, the canonicalization under which their digests were computed, and the full Merkle Root:

$$R_{context} = \text{CatalogMerkleRoot}(\{(h_1, \mathcal{H}(d_1)), (h_2, \mathcal{H}(d_2)), \ldots, (h_n, \mathcal{H}(d_n))\}) \tag{7}$$

Matching full roots, schemes and canonicalizations establish, subject to collision resistance, that every selected handle resolved to the same canonical definition and hence the same external contract. A strict, enforced handshake permits coordination with short references on a match and halts immediately on a mismatch of root, scheme or canonicalization (Figure 3). The third check is needed because two agents can hold identical roots for cards whose pairings are all implied while one hashes under v2 and the other under v3, and only v3 has checked the bindings. Enforcement belongs to the execution harness: it must require and honor the handshake. The guarantee covers *compliant* agents; the protocol alone supplies no Byzantine guarantee. A deployment can additionally require handshake tokens at message admission, rejecting non-compliant agents' messages at that boundary. Integrating these enforcement boundaries is a deployment decision (Section 9). In the release, the Model Context Protocol (MCP) tool `sema_handshake` compares full digests for a single pattern or the roots of a whole vocabulary; for a selected subset, the context tools of Section 6 exchange only an eight-hex-character prefix of its catalog root, which detects drift between cooperating agents but is not a strict check.

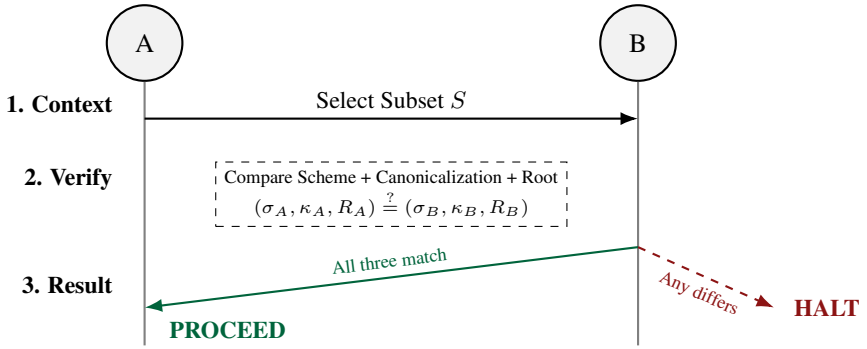

Figure 3: Fail-closed handshake. Agents compare the root scheme σ, the canonicalization κ and the Merkle Root R of the agreed pattern subset; an enforced handshake maps divergence in any of the three to `HALT`.

This represents a deliberate inversion of Postel's Law [20]. 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 choice prioritizes safety over permissive interoperability: compliant deployments reject mismatches in the selected external definitions rather than interpreting them on a best-effort basis. In multi-agent systems where definitional drift can cascade into consequential failure, an explicit fail-closed boundary is valuable.

The four propositions below state what the mechanism provides toward the desiderata of Section 2: Propositions 1 and 3 concern Identity (D1), Proposition 2 Fail-Closed Coordination (D2), and Proposition 4 Compactness and Resolvability (D3, D4). Permissionless Minting (D5) is a property of the design and needs no proposition: any participant can hash a definition (Section 3.3).

Proposition 1 (Canonical Hash Collision Bound)**.** *Under the standard random-oracle approximation, for a vocabulary of size $|\mathcal{V}|$ the probability of two distinct canonical definitions producing the same SHA-256 digest is bounded by:*

$$P(\textit{collision}) \leq \frac{|\mathcal{V}|^2}{2^{257}} \approx 0 \tag{8}$$

This follows from the birthday bound under the stated random-oracle approximation. With $|\mathcal{V}| = 617$, collision probability is negligible ($< 2 \times 10^{-72}$).

Proposition 2 (Strict Fail-Closed Detection)**.** *If agents A_i and A_j resolve distinct canonical definitions d_i and d_j for pattern P, then $\mathcal{H}(d_i) \neq \mathcal{H}(d_j)$ with overwhelming probability. A strict, enforced handshake comparing full digests or full roots therefore halts for compliant agents; cooperative prefix mode provides drift detection only. Adversarial deployments additionally require authenticated references or transport.*

Proposition 3 (External Contract Identity)**.** *If agents A_i and A_j both resolve a full identifier I against an honest content-addressed store, obtain canonical definitions d_i and d_j, and verify $\mathcal{H}(d_i) = \mathcal{H}(d_j) = \text{digest}(I)$ under the same agreed representation and hash scheme, then $d_i = d_j$ subject to hash collision resistance. They therefore hold an identical* external *semantic contract for I, independent of their private internal semantic models M_i, M_j. Sema does not prove $M_i(I) = M_j(I)$; it establishes identity of the externally resolved object under these assumptions, so any divergence in downstream behavior cannot be attributed to the agents having resolved different definitions under that identifier.*

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 $|d|/|r|$. Given a declared resolver or catalogue in which r maps unambiguously to d, expansion is deterministic. Cryptographic assurance still requires verification against the full pattern hash or an agreed context or aggregate root; the short stub is a readable locator, not a self-authenticating security boundary.*

3.6 The Type System

This is the bootstrap library's *particular* type system; other schemas remain possible. Field-level interoperability and resolution require shared schemes, and the current registry and handshake tools implement the Pattern Card case.

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 this schema handles polymorphism, composition, and wiring across 617 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, `ExponentialBackoff#f5e5` with a `base_delay` of 100ms and the same pattern with a 5-second delay share the same Pattern-definition 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 declares a range against which a surrounding runtime may validate instance values while preserving one pattern definition.

To support open-ended reuse without creating incompatible ad hoc interfaces, the Pattern Card schema employs Typed Interfaces. The `dependencies` object partitions declared aliases among four roles: `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. Each alias occurs in exactly one category, making this map the "Single Source of Truth" for dependency wiring. The separate `extends` field records specialization. Acyclicity is enforced by a separate graph check over both dependency and specialization links, yielding a directed acyclic graph (DAG).

Typed interfaces decouple verbs from specific noun instances. (A *verb* here is a pattern that acts, such as `Check#6890`; a *noun* is a pattern that describes data, such as `Task#a850`.) A LogisticsAgent locking a shipping container and a MedicalDrone locking a patient record can both invoke the same pattern when both subjects satisfy the `accepts` interface (e.g., `"accepts": { "subject": "sema:UniqueHandle..." }`). In this example, the agent or runtime must check each subject against the referenced `UniqueHandle` contract; a noun's `data_schema`, when provided, makes its structural constraints explicit. Compatibility depends on the declared contract rather than variable names. This architecture allows the 617 core patterns to support many cross-domain compositions through shared noun definitions.

The integrity of these compositions is enforced by the Explicit Wiring Rule, which welds the abstract type to the concrete logic. The validator 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. An accepted Pattern Card therefore cannot declare a signature without the required links in its hashed specification; this validates the card's wiring, not the behavior of an eventual implementation.

The Pattern Card tooling resolves these high-level intents into concrete pattern handles. When an agent submits a polymorphic intent such as `Deep(Discover)`, the registry resolver looks for that exact declared signature (e.g., on `DeepResearch#2e2b`) and returns the selected card with its dependency subgraph to the requested depth. The current resolver selects the first matching entry in registry iteration order; no match produces a not-found result. Different registries can therefore select different implementations of the same intent. Coordination verifies the resulting concrete full identifiers or context root (Section 3.5), rather than treating the signature alone as a shared definition. This resolution step decouples intent from a specific pattern definition, allowing the system to retarget the underlying procedure, such as swapping a heuristic check for a rigorous crypto-audit, without altering the agent's cognitive script. Executing the resolved procedure still requires an agent or implementation binding.

Table 1: The Deep Registry: Resolving Polymorphic Rigor. Fast and Slow refer to lightweight versus heavyweight resolution strategies. Only `Deep(Discover)` is currently declared, on `DeepResearch#2e2b`; the other rows show intended pairings, which return not-found until a card declares their signatures.

Intent (Query)	**Source Pattern (Fast)**	**Resolved Target (Slow)**
`Deep(Heuristic)`	`HeuristicSnap#d683`	`MentalSim#c28c`
`Deep(Trace)`	`Trace#b68d`	`GenealogicalTrace#67cf`
`Deep(Discover)`	`Discover#f6e0`	`DeepResearch#2e2b`
`Deep(Nature)`	`Nature#9ac1`	`LivedProof#4aa1`

4 The Bootstrap Library

This section presents the Sema Bootstrap Library: the Schelling-point vocabulary introduced in Section 3. The library is an initial seed rather than a closed canon; if an agent requires a different locking mechanism, it can permissionlessly mint a distinct variant such as `LeaseStateLock` without breaking the protocol.

4.1 Methodology

The initial vocabulary is a pragmatic bootstrap assembled without a principled selection procedure. A different author could produce a different vocabulary using the same protocol; the protocol's value is independent of any particular library. The bootstrap library's first 457 patterns emerged iteratively from four sources: distributed-systems primitives, the cognitive-decomposition architecture developed in [21], model-assisted gap analysis, and the author's domain analysis of agent failure modes. Creativity protocols such as adversarial evolutionary generation [22] were used for some frontier patterns. Later additions brought the library to 617 patterns: coined words for concepts that have no English name; processes described without a prescribed response; reasoning patterns, including patterns for evidence and explanation; further cards from the decomposition algorithm of [21]; population and historical dynamics; and procedures for verification, change management, and triage.

Admission to the bootstrap library requires at least one of two curation criteria. A concept earns a pattern if it requires *protocol consistency*— multiple agents must coordinate on the exact semantics, as with `StateLock#c9c2`—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 reviewed through a broad-use test: compare legitimate deployment contexts, retain the mechanism and invariants they share, and place context-specific requirements in descendants via `extends`. For example, `Task#a850` specifies recursive intent and inherited constraints, while `BoundedTask#86dd` adds an explicit budget enclosure and quality gate. The `Lock#95c2` $\rightarrow$ `Mutex#b031` family likewise preserves the general exclusion contract while specifying a concrete ownership mechanism. This review seeks a useful intersection: broad enough to support legitimate variants and concrete enough to constrain behavior. It does not exhaust every possible deployment context.

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 independently of any author (e.g., `Lock#95c2`, `Entropy#d5f7`, `Gradient#ecfc`, `Equilibrium#4f03`, `Conservation#0b32`, `Distance#dfdf`, `PhaseTransition#b775`, `Attractor#0036`, `MutualInformation#e714`, and `Measurement#511d`). Above this rests **Mind**, mechanisms that structurally require cognition (e.g., `BayesUpdate#8656`), 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#01ac`). 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., `Mutex#b031`).

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. Placement follows the mechanism's execution requirements rather than its typical application domain or subject matter. This test is the governing principle recorded in `docs/core/philosophy.md §3.1`.

The four-layer ordering is a design discipline for this bootstrap library; other semantic vocabularies may use different decompositions. Its 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 illustrates the dependency error this discipline rejects.

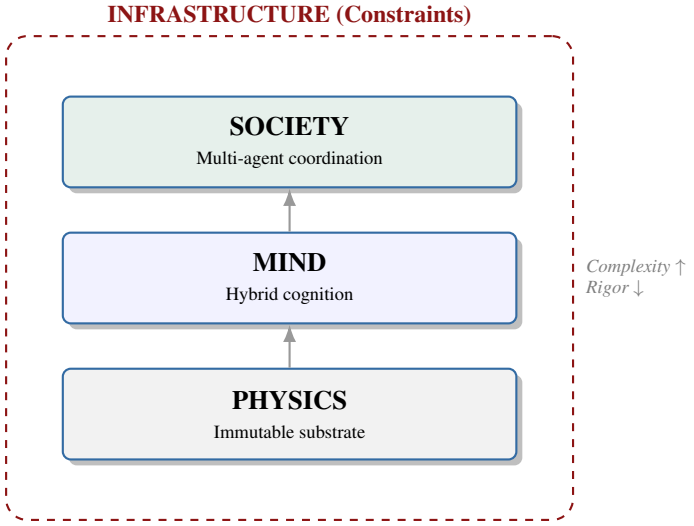

Figure 4: The Civilization Stack. Physics supports Mind, which enables Society; arrows show support from lower to upper layers. Infrastructure encloses this conceptual stack as authored constraints and operations. The bootstrap's separate hard-dependency ordering is Society → Mind → Physics → Infrastructure, with same-layer dependencies also permitted.

This stratification supports a property similar to Stewart Brand's "Pace Layering" [23]: lower layers provide stability for the turbulence of upper layers. Separating **Physics** primitives such as `Entropy#d5f7` from **Society** agreements such as `Constitution#3deb` allows social contracts and voting protocols to change without redefining lower-layer terms. The layer discipline therefore reduces coupling between governance and substrate guarantees.

The dependency graph must form a DAG because recursive dependency hashing requires an acyclic order; the validator rejects cycles. Layer direction is a separate library policy: an acyclic graph can still contain a dependency from a mechanical primitive to a social agreement. Because layer metadata is excluded from the hash, a card's identity depends on its canonical specification and references, while its filing layer remains mutable. Libraries can therefore adopt different layer policies and retain hash-level interoperability when they share the Pattern Card scheme.

Table 2: Bootstrap library: 617 patterns across 14 category names and 4 layers

Layer	Category	N	Examples
Physics	Dynamics	6	Brakefade, Ripplelag, Swayhold
	Primitives	15	Attractor, Causation, Conservation
	Time	1	CausalBarrier
Infrastructure	Data Structures	101	AcceptSpec, Aesthetics, Anomaly
	Dynamics	3	Formelt, Limitblur, Regulapse
	Primitives	54	Act, Actor, Aggregate
	Safety	1	ExpandContractMigration
	Verification	14	AuditTrail, ClosureConservationCheck, CompatibilityCheck
Mind	Dynamics	14	Aimscatter, Capacitylag, Crossripening
	Inference	27	AbsenceEvidence, BaseRateInclude, BayesUpdate
	Memory	18	ArchiveServingFrontier, BeliefTracking, ChunkMerge
	Reasoning	102	Abduction, Almostnot, Analogysnap
	Strategy	105	ActionConvergence, AdversarialSteel, Agent
Society	Coordination	12	Compromise, Consensus, ConsensusFinder
	Dynamics	20	CumulativeCulture, DemographicTransition, DensityRatchet
	Economics	9	AtomicBid, AttentionMarkets, Award
	Governance	24	Absentseat, AnchorDrop, Burdenlight
	Protocols	91	AdversarialProof, AgentDiscover, AgentProtocol
	Total	**617**	

The bootstrap library applies a stricter policy (Section 4.8): hard dependencies (`accepts`, `composes_with`) must remain within the same layer or flow to a more fundamental layer (Society → Mind → Physics → Infrastructure), 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#09e1` in Infrastructure soft-references `Agent#e127` in Mind because beliefs are conceptually held by agents, and the reference is acyclic). The current vocabulary contains 617 patterns organized under 14 category names across these four layers (Table 2; complete taxonomy in Appendix B), with zero patterns violating the layer-direction rule at this version.

Each layer also has a Dynamics category for patterns that describe how a system behaves and prescribe no response, so a card filed there reads, even on its own, as a model to recognise in a case rather than steps to follow; the layer still follows what the mechanism requires. Section 4.7 describes one, `RegressionToMean#9c60`.

4.3 Contract Formalism Tiers

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

Tier 1 (Ironclad) marks the strongest specified contracts (invariants and, where they apply, preconditions and post-conditions), intended for autonomous use where an implementation or harness checks the relevant clauses; 319 patterns carry it.

Tier 2 (Honesty-Dependent) includes patterns with clear mechanisms that assume alignment-seeking agents. These are intended for cooperative contexts; adversarial use requires stronger alternatives and enforcement due to collusion risks (e.g., `SteelmanCheck#b004`).

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

The tier is metadata that an author assigns and review can change; it is not computed from a card's fields, whose coverage Table 4 gives. Tiers 2 and 3 are carried by 283 and 6 patterns, and 9 primitives of ring 0, such as `Actor#86fc`, carry a tier 0 that marks a primitive rather than rating a contract.

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.

4.4 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. *Primitives* act as the fundamental building blocks of agent cognition. Five foundational verbs map to dimensions of autonomous inquiry: `Discover#f6e0` (Breadth) acts as the Scout, querying the horizon to find new nodes, agents, or paths; `Deep#16b0` (Depth) serves as the Scientist, escalating from heuristic proxies to rigorous implementations; `Trace#b68d` (Time) functions as the Historian, reconstructing causal chains within the records it holds; `Check#6890` (Validity) operates as the Critic, filtering inputs against invariants; and `Stigmergy#dd4f` (Space) works as the Builder, writing persistent signals for indirect coordination. Alongside these verbs, noun primitives such as `Belief#09e1`, `Agent#e127`, and `Constraint#7013` provide the objects of agency.

These primitives enable a higher-order grammar where complex behaviors compose from atomic units: `Check#6890`(Condition) evaluates a target `Condition#b480` and yields Verified, Falsified, or Unknown without changing control flow; `Trace#b68d`(Target) wraps a process to generate an immutable lineage log whose coverage may be partial; and `Stigmergy#dd4f`(Signal) marks the shared environment with a signal for others to discover.

Macros are composite patterns built from primitives via semantic import. These grammatical roles describe how patterns compose; they are separate from the library's mutable ring assignments and its safety-rigor tiers (Tier 1–3, Section 4.3). For example, `AgentDiscover#b784` imports `{{Discover#f6e0}}` acting on `{{Agent#e127}}`, while `TraceBelief#5da5` imports `{{Trace#b68d}}` acting on `{{Belief#09e1}}`. This grammar allows agents to express intent via queries like "I need to `Deep(Trace)` this belief," which the resolution workflow described in Section 3.6 can connect to a concrete pattern definition once a card declares that signature.

This distinction reveals a bicameral architecture. Primitives serve as the query interface, allowing agents to identify logical gaps in the vocabulary. Macros like `IdentityHandshake#4397` serve as the social interface, enabling efficient coordination through compression. A verified Macro identity commits to its imported components, so agents can establish one composite agreement covering several primitives. This can reduce exchanges in workflows that would otherwise negotiate those components separately. Primitives are for querying; Macros are for talking.

The verb/noun distinction used in the Type System (Section 3.6) also addresses the "schema drift" problem where implicit schemas lead to subtle incompatibilities. Pattern Cards formalize the "shape of work" as content-addressed Data Patterns. `Task#a850` represents a recursive definition of intent that inherits constraints from its parent and requires acceptance criteria in its data schema. `Solution#8d55` acts as a container for work product, encapsulating provenance and, for composite work, a component tree for supply-chain auditing. `UniqueHandle#1853` specifies a rivalrous resource pointer under linear logic: a conforming runtime treats transfer from Agent A to Agent B as revoking Agent A's access. To ensure structural compatibility, Noun definitions may include a validation schema in the `data_schema` field (e.g., `Task#a850`, `Bid#647f`), embedding constraints directly into the identity unlike external standards such as SHACL [24]. This allows agents to construct rigorous semantic sentences:

> *"I `Delegate#00fa` this `Task#a850` to you, expecting a `Solution#8d55` that satisfies this `AcceptSpec#37fa`."*

4.5 Cognitive Architecture: The Solver Stack

Each Solver node decides whether to execute the work directly or to decompose it and delegate to child Solvers. The vocabulary encodes a 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 [21]. The vocabulary provides the content-addressed layer where each concept from the Composable Reasoning Protocol becomes a pattern with machine-verifiable identity.

The abstract `Solver#a447` contract is an interface any `Agent#e127` can take on for the duration of a Task. `PolymorphicSolver#ce9a` specifies its default realization within the `UniversalSolverTree#f9fb`, across substrates such as an LLM, a human, a hybrid, a tool-using agent, or a nested composition. Each Solver exposes the same five surfaces: Manifest (`Card#e4b2`, a structured advertisement of what the Solver can do, distinct from a Pattern Card), Execute (`PolymorphicSolver#ce9a` itself—the universal atom), Consult (`SocraticLoop#ffaa`), Verify (`Validate #ffd1`), and Feedback (`PerformanceSignal#9dfb`). The Pattern Cards specify these contracts; an agent or executable binding realizes them.

`SolverDecomposition#ff4b`, the solver-bound kind of `ConceptualDecomposition#f1f4`, makes the recursion contract-bound: each sub-concept exposes the same Solver interface and is therefore delegatable, a stronger requirement than generic `Decompose#6994`. Inputs arrive as a `Task#a850` with all inherited constraints, and outputs return as a `Solution#8d55` carrying provenance, including the component tree for composite work. `ComputeBudget#4091` provides a pre-execution gate, while `MarginalValueRule#6611` governs whether another level of decomposition is worth its cost. This uniform interface makes a composed solver usable as a node in a larger composition. The claim that typed composition can extend reasoning beyond the capacity of an individual model is developed in [21].

4.6 The Babbage Principle of Cognition

The vocabulary applies the *Babbage Principle* [25] to artificial intelligence as a routing objective: assign each subtask to the cheapest competent solver. The uniform Solver contract gives this division of cognitive labor a common interface. In the `FractalIntelligence#7739` architecture, `Decompose#6994` separates work into sub-tasks that can be handled by specialized, lower-cost agents or deterministic scripts, reserving more capable solvers for work that requires them. The budget and marginal-value rules connect this routing objective to the cost of further decomposition. Recent empirical work on atomic "micro-agent" decomposition [26] and DeepMind's AlphaEvolve [27] provide related support for decomposition-based routing. See [21] for the full treatment.

4.7 Example Patterns

Selected Pattern Cards illustrate how mechanisms, contracts, and failure modes work together across the library's tiers. `StateLock#c9c2` (Protocols) specifies atomic coordination via temporary state fusion: two agents fuse a subset of writable state, and changes require both signatures. Fusion ends on completion or timeout; premature dissolution while legitimate work remains is one documented failure mode. `SpectralTune#c0aa` (Protocols) compares content-addressed ontology references before data transfer. The sender transmits hash-based tuning signals and the receiver checks the referenced context. Its current contract requires halt and escalation on mismatch, exposing disagreement while risking an over-strict halt on benign version or encoding differences. The negotiation-loop failure that motivated this rule is described in Section 7.5.

For cognitive safety, `SteelmanCheck#b004` (Reasoning) mandates a claim-specific counter-argument, uses `Judge #704c` to determine whether it qualifies as a steelman, and then uses `Check#6890` to test whether the conclusion remains robust; however, same-agent grading remains vulnerable to sycophantic opposition. Finally, `WhyClimb#ed20` (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.

Not every card prescribes steps. `RegressionToMean#9c60` (Dynamics) describes a process: a result picked out because it was extreme is usually followed by one closer to typical, because part of the extremity was chance. An agent tests such a card against a case, and its failure modes name the misreadings to check for, such as crediting the whole return to the response the extreme prompted.

4.8 Refinement

The vocabulary evolves through new mints, retirements, mechanism rewrites, and relocations. Refinement is the cross-pattern stage of this process: an end-to-end review against the current governing rules. With hundreds of patterns in place, layer coherence, redundancy, and principle consistency require systematic comparison across the library in addition to checks performed on individual cards at mint time.

The process became more structured as the library grew. Refinement checkpoints used a dedicated "Taxonomist" role to reject duplicates, merge overlapping concepts, and enforce the Noun/Verb separation; repeated failures became rules and guidelines. Structural validation is required before `sema apply` mutates the vocabulary; Section 6 details those gates. Separate corpus audits report similarity, redundancy, and contract-field coverage as advisory evidence for Taxonomist review and human adjudication.

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 durable requirement is auditability: each accepted change states the rule it depends on, the failure mode it prevents, and why the alternative was rejected.

`Bid#647f` illustrates a retained-placement decision. Its initial Society classification followed social-economic intuition; review retained that placement for a structural reason: a binding offer runs from a solver to a task-assigner, so its mechanism requires a counterparty. Treating Bid solely as a data artifact would miss that execution requirement. The example shows how the mechanism-sufficiency test can confirm an existing placement as well as motivate a relocation.

A second class of refinement decision concerns *scope* rather than *placement*: a pattern may be coherent, well-mechanized, and correctly layered, yet still be inappropriate for the default library that downstream consumers install. During an earlier 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 separated from the default library into an experimental database. That database is now retained only as a frozen historical snapshot; current tooling does not select, package, or treat its contents as an active library. Such scope decisions are made per pattern on the basis of how the mechanism composes with realistic deployment contexts rather than on whether the mechanism is well-formed, which the other gates assess separately. The current distribution architecture is described in Section 4.9.

After the refinement pass, the applied audits detect no duplicate mechanism strings, canonical-form duplicates, or hash collisions across 617 patterns. Embedding-similarity sweeps provide a further screen for near-duplicates (Table 5). These audits and Taxonomist review support library quality, but they do not guarantee functional uniqueness (see §9, "The Synonymy Limit"). We make no claim to completeness or optimality.

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 maintained canonical vocabulary supplies the default library, while each consumer keeps a local database. Consumers may delete, decline, or pin upstream patterns and add local extensions. A *pull* operation verifies incoming definitions against their published digests and synchronizes along the dependency graph, integrating a pattern only after its dependencies are verified locally.

The local semantic-set and catalog roots identify the database's current contents, and an exclusion file records upstream patterns the consumer has declined. Unhashed successor metadata supports migration after retirements or renames, without redirecting references or archiving old content. The current database stores one active definition per handle; historical references remain resolvable only while a store retains the corresponding content.

Refinements become available at the next pull, allowing consumers to combine adopted changes with retained upstream versions and local extensions. A local vocabulary therefore need not correspond to one canonical release. The library behaves as a living commons: definitions sharpen over time, safety boundaries stay explicit, and divergence between consumers remains visible through their content-addressed snapshots.

The centralized-canonical model is one governance path among several supported by this architecture, not a constraint imposed by Sema's communication primitive. Fully decentralized variants are compatible with the same hashing and

[3] `https://github.com/emergent-wisdom/sema/blob/v0.11.0/docs/manuals/vocabulary-design.md`

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's identity derives from its hashed canonical content, not from who hosts it, so two consumers that verify the same full hash resolve the same canonical definition regardless of which governance structure produced it. Coexisting libraries are already implemented (Section 6): a library installs from its published manifest only after verification and cannot supply executable or prebuilt state, and a public registry lists published libraries. The bootstrap uses a canonical-pull model to minimize onboarding friction and give the refinement methodology a clear locus; the same communication primitive remains compatible with the stronger decentralization requirement identified in the earlier Living Taxonomy work [7].

5 The Living Taxonomy of Thought

The static properties described so far (hashing, types, and vocabulary structure) support a dynamic conceptual layer in which agents can reuse prior distinctions, identify meaningful departures from them, and leave those departures behind as new cognitive infrastructure. This architecture builds on the "Living Taxonomy of Thought" sketched in earlier work [7] as a component for self-improving AI. The following architectural proposals build on the implemented Pattern Card machinery; Section 6 records the status of each component.

5.1 The Genesis Loop

The proposed *Genesis Loop* is a recursive process in which an agent proposes and revises vocabulary patterns. Because patterns encode reusable procedures and models of how systems behave, rather than facts alone, the proposal concerns the repertoire of explicit reasoning operations and the models they apply, not only the accumulation of factual content. It is related to LLM self-evolution frameworks based on iterative refinement and self-correction [28], and to the *Nested Learning* paradigm [29], where rapid inner-loop adaptation is distinct from slower outer-loop consolidation.

The loop would proceed in four phases. First, in *Hypothesize (Generativity)*, an agent detects friction or explores latent space (via `ConceptBlend#4e1a`) to generate a candidate mechanism. Second, during *Judge (Evaluative Merit)*, `Judge#704c` would assess the candidate's structural merit on a scalar scale. Third, *Harden (Adversarial Evolution)* would subject the pattern to `AdversarialSteel#90b1`, where a "Devil's Advocate" attempts to exploit its invariants. Finally, in *Mint (Crystallization)*, an accepted candidate is validated and hashed, becoming an immutable pattern available for citation and reuse through `sema_mint`.

The Hypothesize phase can draw candidates from observed coordination failures, systematic refinement passes, imported domain mechanisms, or lexical exploration. These sources differ in where an idea comes from. Within the proposed loop, each candidate would encounter the same downstream stages—Judge, Harden, Mint—so diverse ideation would feed a common review process. Participating libraries could retain their own acceptance policies while sharing the resulting content-addressed patterns.

Documents can supply such candidates too. An agent can read through a body of papers, manuals or interview transcripts, pull out the mechanisms they describe, reuse an existing card wherever one already captures a mechanism, and draft cards for the rest. The released search, validation and minting tools support each vocabulary step of that work. The drafts then meet the same review as any other candidate, and those that pass can join an existing vocabulary or form a library of their own (Section 4.9).

The bootstrap library and earlier work on ontological discovery [22] 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 four-layer hierarchy and typed Pattern Cards impose analogous structure on coordination primitives. The Pattern Card schema leaves taxonomy placement outside identity—metadata remains unhashed—while content-addressing the specification produced by that categorization. If categorization pressure changes a pattern's mechanism, invariants, or dependencies, the hash changes; if only the library filing changes, identity remains stable.

5.2 Evaluation as a Primitive: The Judge

To enable the Genesis Loop, we introduce a new Ring 0 primitive: `Judge#704c`.

While `Check#6890` queries *Validity*, yielding Verified, Falsified, or Unknown, `Judge#704c` is specified to query *Merit* on a scale from 0.0 to 1.0. For a vocabulary candidate, the proposed assessment concerns structural properties such as connectivity and orthogonality. A declared scoring function would make the evaluation criteria explicit. This differs

from assigning confidence to a factual claim, as in claim-analysis systems [30, 31]. Consumers could use the scores as uptake thresholds, allowing higher-scoring patterns to propagate more readily. Defining and calibrating the scoring function is part of the proposed Judge implementation.

5.3 Phylogeny: Specialization and Version Lineage

A living taxonomy needs to represent two different relationships. Both appeared with the Pattern Card (Section 3.2). The hashed `extends` field records specialization: a child claims to be a narrower kind of one exact, immutable parent definition, and the graph exposes that claim as an `IS_A` edge. The unhashed `_meta.supersedes` list records version lineage: the current card replaces one or more earlier identifiers. The first relation is part of what a card means; the second records how that meaning changed over time.

```
{
  "handle": "BoundedTask",
  "extends": "sema:Task#mh:SHA-256:
      a85010f508f85055170c396c8fa390125c1206888c565ac10651f3969a339f8d",
  "mechanism": "A specialized Task enforcing budget...",
  "dependencies": { ... },
  "_meta": {
    "supersedes": ["sema:BoundedTask#mh:SHA-256:..."]
  }
}
```

The implementation stores and queries the specialization graph, while `supersedes` is currently consumed by the pull reconciliation path. A general query such as "show the evolutionary path through every published version," or analytics identifying stagnant and rapidly speciating lineages, requires a dedicated traversal over the replacement records and remains future work.

5.4 The Experience Matrix: From Language to Telepathy

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 resolve the same full `sema_id` against an honest content-addressed store and verify the returned object, they establish that they hold the same canonical definition. This provides verifiable definition identity, not proof of identical internal meaning.

Contextual Telepathy (proposed) Agents hash their recorded conversational state—the accumulated context window at a checkpoint—producing a content-addressed context snapshot. A third agent joining a swarm mid-session could retrieve and restore that snapshot, then verify "I hold the same recorded context you held at step 12." Restoring the snapshot would avoid replaying the interaction sequence that produced it. This would enable resumption, forking, and state-identity verification across agent handoffs: telepathy of *shared recorded context*, with shared understanding remaining a separate question.

Experiential Telepathy (proposed) Agents would serialize learned state (e.g., Q-tables, LoRA adapters, or success-probability tensors) into content-addressed *Matrix Patterns* specified by `ExperienceSharding#50ff`. A text definition explains how to do a task; a Matrix Pattern would carry state learned from having done it. If Agent A optimizes a specific `ExploreExploit#218c` strategy after 10,000 trials, it could mint a heuristic matrix. An Agent B with a compatible state representation, model architecture or base, and task could import the matrix and interpolate it with its own priors, potentially transferring task-relevant learned state: telepathy of *acquired skill*. The required compatibility depends on the artifact: a Q-table needs agreed state and action coordinates, while a LoRA adapter needs a compatible base model. This echoes the *Nested Learning* paradigm [29], where rapid inner-loop adaptation is distinct from stable outer-loop consolidation.

These levels describe a proposed progression from words to shared context and acquired skill. In the envisioned architecture, agreed patterns supply the terms used to describe work, context snapshots carry a recorded stage of that work, and compatible Matrix Patterns carry learning obtained from it. Each uses content addressing at a different granularity: shared-definition identity, shared context-state identity, and potentially transferable learned state.

6 Implementation

The Sema system is implemented as an open-source Python package, `semahash`. Release-specific line counts, tool inventories, and graph-store type catalogues are maintained in the repository.[4]

Three implementation boundaries delimit the evaluation in Section 7. *Implemented and enforced locally*: JSON canonicalization and hashing, Pattern Card schema validation, preflight-validated batch mutation, and dependency graph constraints. *Implemented but deployment-optional*: the MCP handshake (including Semantic Telepathy at the definition-identity level), hybrid search, embedding navigation, dependency expansion, browser tooling, specialization-graph queries, pull-mode synchronization using replacement records, verified installation of published libraries with a public registry, and the Taxonomist's human and model-assisted refinement workflow. *Proposed extensions*: the `Judge #704c` scoring implementation, an automated Genesis Loop, general version-lineage traversal and analytics, Contextual Telepathy, and the Experience Matrix, including tensor serialization and gradient transfer. Section 5 describes how these extensions would use the existing vocabulary machinery; Section 9 identifies the remaining research work.

Schema Validation. Every pattern passes through a Pydantic validation pipeline 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. Section 4 describes the corresponding library-refinement methodology.

Hash Computation. The Pattern Card implementation uses SHA-256 and canonicalization v3 to commit to the eleven semantic fields specified in Section 3. It keeps each dependency alias paired with its target. Each library release and each store database records the canonicalization its identities use, so identities published under v2 remain verifiable, and a pull from a v3 library moves a v2 database to v3 and recomputes its local identities in one step. For backward verification, a pre-0.4 card carrying the former `derived_from` key is hashed under that original key and is not reinterpreted as an `IS_A` claim; explicit migration to `extends` produces a new identity. Dependency and specialization references carry their targets' hashes, forming a Merkle DAG. Retargeting a hashed reference changes the identity of the card, and re-minting carries the change to every card that depends on it, directly or transitively (Section 7 measures how far). Specialized descendants are not carried along: a child pins its parent's exact identity and changes only when explicitly retargeted, as the next paragraph describes.

Batch Operations. The CLI provides a `sema apply` command that validates a batch of additions, removals, and their projected dependency graph before any mutation, then applies the batch step by step, without rollback if a step fails. Preflight rejects missing targets, dangling references, cycles, layer-direction violations, and unavailable exact specialization pins. Additions are topologically sorted using Kahn's algorithm; cycle reports identify the offending path. Because the workspace stores one active definition per handle, a parent replacement must preserve the availability of every `extends` target. Reviewed child cards may be staged and retargeted through an explicit option; no unstaged specialization claim is rewritten. A `--check` flag enables dry-run validation without mutation.

Agent Integration. A Model Context Protocol (MCP) server exposes the Pattern Card operations to LLM agents over stdio: search (hybrid keyword and semantic), resolve (recursive dependency expansion), handshake (hash comparison for a single pattern, or for the semantic-set or catalog root of an entire vocabulary), 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. Given a pattern reference and a remote hash, the handshake tool returns `PROCEED`, `HALT`, `PROVIDE_HASH`, or `REQUIRE_FULL_HASH`. In the default cooperative mode, a matching short prefix may produce `PROCEED` with prefix assurance; in strict mode, only a full 64-character match proceeds, while a matching prefix produces `REQUIRE_FULL_HASH`. Each answer also states the canonicalization under which the vocabulary's digests were computed, and strict mode requires the peer's to match: a card whose pairings are all implied has the same digest under v2 and v3, but only v3 has checked the bindings, and a peer that declares no canonicalization is taken to use v2.

The current convenience tools `sema_propose_context` and `sema_verify_context` exchange an eight-hex-character catalog-root prefix and label it explicitly as cooperative drift detection, not adversarial security; they require the declared

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

scheme even in this mode. A full semantic-set root answers the weaker question of whether the raw definition-digest sets match; those digests can still change when target handles in structured references are renamed.

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. Keyword matching weights the field matched (handle: 1.0, signature: 0.75, gloss: 0.70, mechanism: 0.55) and scores both the whole query and each query word; rarer words count more, and a single word matching a handle counts 0.9. Reference implementations exist for a small set of patterns (SpectralTune, StateLock, ProphetFanOut, CounterfactualAnchor are illustrative); the rest of the library ships as Pattern Cards without executable bindings, by design—the tooling identifies definitions, not running code.

Deployment Paths. Context hydration—resolving a Pattern Card and placing its definition in model context—is the deployment path evaluated in this paper. The MCP server implements resolution and returns the Pattern Card data; the host or agent workflow decides whether and how to place it in a system prompt or other context. A weight-resident path (mixing references like `StateLock#c9c2` directly into training corpora, so learned associations with pattern references reside in the model's weights rather than the context window) is also compatible with the communication primitive. We call such a system a *Sema-native model*: a model that has learned a vocabulary well enough to use and interpret its content-addressed terms naturally in language. Section 8.5 gives the training proposal this path comes from. Intermediate architectures (fine-tuning on a library, retrieval-coupled models, hybrid hot-patterns-in-weights) are likewise possible. Hydration is model-agnostic and requires no training infrastructure, but it is not automatic in the current tooling. A deployment that resolves and verifies a full identifier can check at inference time that the supplied definition matches its digest, while a weight-resident deployment cannot make the same claim from weights alone. Closing that gap (signed training manifests; audit-coupled retrieval) is an open deployment-architecture question.

Agent Skills. An Agent Skill [32] is a directory whose `SKILL.md` file gives a name, a description, and instructions that an agent loads when a task calls for them. The repository ships four skills of this kind, which teach an agent how to use the tools described here. The reverse is also possible, though nothing in the release does it: a skill can be carried inside Sema. A skill is a named procedure, as a Pattern Card is, and there are two ways to carry one. The first leaves the skill as written and hashes its files under an agreed representation (Section 3.1). Its name then becomes a verifiable word: an agent asked to use a skill by its full reference can check that it holds the instructions the other agent meant, and a skill that builds on another can pin it by hash. The skill format defines no field for a skill's exact content and no way for one skill to depend on another; its free-form `metadata` map could hold both. The second writes the procedure as Pattern Cards: the instructions become mechanisms, the checks become invariants and failure modes, and the steps refer to existing cards. The skill is then a small library, installed and verified by the released library tooling (Section 4.9) and loaded by the context hydration described above. In both cases a matching hash shows which skill an agent holds, not that the skill is sound or safe to run (Section 9).

6.1 Interactive Tooling

The vocabulary is deployed as an open-source web application.[5] 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 617 patterns by the four Civilization Stack layers (Section 4). Real-time hybrid search combines keyword matching with embedding-based semantic retrieval, allowing queries like "consensus" to surface `LazyConsensus#76f5` alongside structurally related patterns such as `LatticeCommit#488a`.

Taxonomy Graph. An interactive 3D force-directed graph renders the stored taxonomy paths, patterns, indexed contract clauses, and their relationships. This view exposes the implemented graph, while Table 3 separately describes a projection from all measured card fields.

Programmatic Access. The command-line interface supports preflight-validated 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.

[5] Source: `https://github.com/emergent-wisdom/sema`. The site at `https://semahash.org` is a separate hosted deployment built on the released package.

7 Analysis and Validation

This section separates evidence by layer. Hash determinism exercises the JSON/Merkle implementation; graph structure, embedding-based distinctness, token compression, and adversarial refinement characterize the Pattern Card bootstrap library; and the agent demonstrations exercise the handshake and surrounding tooling. Together the subsections cover six dimensions, but the library and demonstration results are not general evidence for every representation that could implement Sema's communication primitive.

7.1 Knowledge Graph Structure

The bootstrap library has a substantial relational structure, summarized in Table 3 as a *field projection*.[6]

Table 3: Pattern Card field-projection statistics

Component	Count
Projected nodes	4,427
Projected edges	6,114
Patterns	617
Unique mechanism texts	617
Invariants	1430
Unique pre/postcondition clauses	554
Unique failure-mode clauses	1209
Projected edges per pattern	9.9

The projection makes explicit how each card relates its mechanism to contract clauses and other patterns.

Exact dependency commitments have a cost: editing a card's hashed fields gives it, and every card that depends on it directly or transitively, a new identity. The cost is uneven: 268 cards (43%) have no dependents, while 66 have a hundred or more, and an edit to `System#f8eb` would change 421 other identities, 68% of the library. The median Infrastructure card has 54 dependents; the median Mind or Society card has none. In one earlier release, repairing eight definitions and retargeting two specializations changed 267 of the then 457 identities, 257 of them only through a changed dependency. Hub corrections are therefore batched into one release, and each changed card lists its earlier identities in the unhashed `_meta.supersedes` field: 443 cards list 1,464, none of whose content the current store holds (Section 9).

7.2 Hash Stability and Contract Coverage

We first verify deterministic minting across the full vocabulary through two checks. Roundtrip integrity ensures that for all 617 patterns, $\text{parse}(\text{serialize}(p)) = p$. Hash determinism confirms that re-exporting the vocabulary produces identical hashes (617/617).

We next measure the library's contract-field coverage and exact mechanism-text uniqueness without paraphrase normalization (Table 4). These checks establish how much explicit structure the cards contain and whether mechanism strings repeat; functional distinctness requires a separate semantic and conceptual review.

Invariants are close to universal and the other clause kinds are not. That follows from the broad-use test of Section 4.1: a general card keeps the mechanism and invariants that every legitimate context shares, and conditions that hold only in some contexts belong to its descendants. A low share of preconditions or typed interfaces is therefore not a count of unfinished cards.

[6]This projection counts one pattern node and one mechanism-field node per card, plus invariant, precondition, postcondition, and failure-mode strings deduplicated within each field after trimming and lowercasing. Its edges count dependency and related links, mechanism links, and clause occurrences; taxonomy-path nodes are excluded. These are descriptive counts derived from the cards, distinct from the SQLite store's indexed and merged nodes. The implemented graph indexes patterns, taxonomy paths, invariants, and pre/postconditions; mechanism and failure-mode text remain available on the cards without separate stored nodes. Current-library inputs: `data/vocabulary/` and `data/taxonomy.db`, retained in the Sema 0.11.0 release snapshot; generation script: `scripts/calculate_graph_stats.py`, in the accompanying repository. The embedding analysis below reuses the bundled cached pattern vectors.

Table 4: Contract-field coverage and exact mechanism-text uniqueness

Metric	Value
Total patterns	617
Unique mechanism texts	617
Duplicate nonempty mechanism texts	0
Patterns with invariant clauses	577 (94%)
Patterns with preconditions	182 (29%)
Patterns with postconditions	172 (28%)
Patterns with typed I/O	81 (13%)
Patterns with parameters	101 (16%)
Total parameters	193
Patterns that compose	94 (15%)

7.3 Semantic Embedding Analysis

To probe vocabulary spread beyond exact-string comparisons, we computed pairwise cosine similarities between cached pattern embeddings (384-dimensional vectors from `all-MiniLM-L6-v2`). The graph store constructs each embedding from the handle, gloss, and mechanism. Table 5 reports all unordered pairs, including negative similarities.

Table 5: Pattern embedding similarity distribution (handle, gloss, mechanism)

Similarity Range	Pairs	Percentage
$[-1.00, 0.00)$	4,167	2.19%
$[0.00, 0.30)$	158,814	83.57%
$[0.30, 0.50)$	26,173	13.77%
$[0.50, 0.70)$	857	0.45%
$[0.70, 1.00]$	25	0.01%
Total pairs	**190,036**	

Key statistics:

- Mean similarity: 0.19, indicating low average cosine similarity under this embedding representation.
- Maximum similarity: 0.83, below the workflow's operational near-duplicate screening threshold of 0.92.
- **25 pairs at or above 0.70** (0.01%). Examples include `Skeleton#d950` / `SkeletonOfThought#546f`, `TreeOfThoughts#8b8c` / `GraphOfThought#220a`, and `FailureTrace#8085` / `ReceptivityGate#0db5`: related procedures that merit comparison during curation.

Together, exact-string uniqueness and the low embedding similarities support a library of varied textual specifications. The screening threshold is an operational curation heuristic, not a calibrated test of functional equivalence. Functional synonymy and conceptual overlap remain questions for manual review and the Taxonomist workflow (see §9, "The Synonymy Limit").

7.4 Token Compression

We measure wire-reference compression using the `cl100k_base` tokenizer, comparing a short `Handle#stub` reference with five contract fields—mechanism, invariants, preconditions, postconditions, and failure modes—across illustrative patterns (Table 6). The measured contract text excludes the remaining canonical Pattern Card fields and the costs of resolution and full-identifier verification.

Table 6: Five-field contract text versus short wire references (cl100k_base tokens)

Pattern	Ref (tokens)	Contract (tokens)	Ratio
`StateLock#c9c2`	6	135	22.5×
`SpectralTune#c0aa`	7	140	20.0×
`BayesUpdate#8656`	6	211	35.2×
`SteelmanCheck#b004`	6	342	57.0×
Average	6.25	207.0	**33.1**×

Patterns with longer mechanisms and more contract clauses achieve higher measured compression. The ratio applies to wire transmission only; using a reference still requires an agent or host runtime to resolve and place the underlying definition in model context (Section 3.1).

Library-wide Compression by Layer. Averaging the same five measured fields 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 measured contract text is longer (judgment criteria, adversarial failure modes, multi-party invariants). The library-wide value of 26.7× across 617 patterns is the ratio of mean contract-text tokens to mean short-reference tokens, equivalently the ratio of their corpus totals. It is not the unweighted mean of per-pattern ratios. The layer averages suggest that deployments drawing more heavily on Society or Mind patterns could encounter higher wire-reference ratios than deployments dominated by Infrastructure patterns; actual savings depend on which patterns recur and how definitions are cached.

Table 7: Compression by layer: mean cl100k_base tokens per pattern; Ratio divides the two means

Layer	N	Ref	Contract	Ratio
Physics	22	5.8	173.6	30.1×
Infrastructure	173	5.2	129.1	24.7×
Mind	266	6.4	172.3	27.1×
Society	156	6.3	174.1	27.4×
Library	617	**6.0**	**160.7**	**26.7**×

7.5 Adversarial Hardening

Beyond static analysis, we reviewed the vocabulary through a "Devil's Advocate" methodology: an agent attempted to exploit pattern assumptions, motivating the five contract and dependency repairs below. Turning their behavioral requirements into operational defenses requires runtime enforcement.

Blinking Light Livelock (Physics) A malicious agent spams lock requests on `StateLock#c9c2`, 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#9e59` and `Cooldown#c288` via explicit dependencies, requiring a selected delay policy and cooldown period between lock acquisitions without forcing every StateLock deployment to use exponential growth. The composition alone does not guarantee fair access.

Blind Search Infinite Loop (Society) When `SpectralTune#c0aa` 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: The current card states the fail-fast requirement: "On hash mismatch, halt and escalate rather than retry." Escalation to `OntologyHandshake#a43c` or human review separates reconciliation from repeated tuning. The resulting trade-off is explicit: a benign version difference can halt a channel even when the parties might otherwise understand one another.

Strawman Waltz Collusion (Mind) In `SteelmanCheck#b004`, 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 remains Tier 2 because same-agent grading can collude. Its specified contract uses `Judge#704c` to score whether the counter-argument qualifies as a steelman, and `Check#6890` to evaluate separately whether the conclusion remains robust; adversarial contexts use `AdversarialSteel#90b1`. Runtime enforcement of Judge scoring is not implemented.

Trojan Horse Sandbox Escape (Infrastructure) An agent wraps a prohibited action (e.g., network access) inside an opaque `ToolInvoke` call, hoping the outer `Task#a850` constraints don't propagate into the tool's execution context.

Defense: The Task contract carries inherited constraints across tool calls, sub-agents, and delegations, while `ToolInvoke#1096` requires the tool execution context to remain within the Task's permission boundary. This is the *Holographic Inheritance* intuition: encapsulation must not erase the restrictions on the work it encloses.

The Semantic Inversion Attack (Topology) An adversary (or a careless architect) creates a valid DAG that is semantically inverted, such as making `Deduction#b9a0` depend on `Vote#6ba4`. If Deduction actually invokes Vote to determine whether an inference holds, its reasoning procedure now depends on a social decision. A valid dependency graph alone does not exclude this mismatch between a pattern's stated role and its required mechanism.

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 two strategies below resolved 67 of the most critical, and the refinement pass described in Section 4.8 resolved the remainder by sharpening the layer definitions (the mechanism-sufficiency test, §4.2) 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#e127` and `Act#2dfe` by splitting the concept. We created `Actor#86fc` (Infrastructure) as the "capability container" that executes acts, while `Agent#e127` (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 across layers 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. Where a card's text does name a pattern in another layer without invoking it, the link is a hashed `references` dependency, which the layer rule exempts (Section 4.2).

These attacks shaped the bootstrap library's four-layer organization of contracts and dependencies. In the subject-matter reading the library began with, which the labels on the attacks above follow, Physics grouped resource constraints, Society grouped coordination mechanisms, Mind grouped cognitive safeguards, and Infrastructure grouped containment and operational primitives; the mechanism-sufficiency test (Section 4.2) has since replaced that reading as the rule of placement. Patterns that assume honest agents are now explicitly marked Tier 2.

7.6 Protocol Integrity Verification: The Babel Test

To exercise handshake-mediated coordination in a live setting, we conducted an agent-in-the-loop demonstration with **Gemini 3.0 Flash Preview**. An autonomous agent equipped with the `sema_handshake` tool was presented with two scenarios: a valid four-character pattern stub and a counterfeit value mimicking a modified definition. The available harness instructs the agent to call the tool and follow its verdict, uses automatic function calling, and classifies the final response heuristically.[7]

In the valid case, the agent reported that the cooperative prefix comparison returned `PROCEED` and accepted the proposed reference. In the counterfeit case it reported `HALT` and rejected coordination. The following excerpt records the two responses; the harness labels each reply by its scenario, so "attacker" marks the agent's answer to the counterfeit:

```
[21:06:41] verifier:  ‘‘I have verified the semantic alignment for the Abduction#e738
pattern.  The handshake returned PROCEED, so I agree to use this pattern.’’
```

[7] Available demonstration harness: `experiments/demos/agent_handshake.py`. The historical excerpt below is retained as reported; the complete original tool-event record and matching `Abduction` snapshot are not supplied with this harness. The separate multi-agent vocabulary snapshot cited below carries a different Abduction version.

```
[21:06:44] attacker: ‘‘The handshake for Abduction#e738 with hash deadbeef returned
HALT due to semantic drift. Therefore, I must reject the coordination.’’
```

The model's phrase "verified the semantic alignment" exceeds what a four-character comparison establishes: prefix matching, rather than semantic alignment or full-hash identity. Within this instructed scenario, the agent's replies followed the reported tool verdicts, giving the workflow an explicit checkpoint for definition agreement. The response excerpt does not establish a causal effect on the agent's decision or enforcement at a downstream execution boundary.

Verification Complexity and Latency. Comparing two cached, fixed-size digests is $O(1)$ with respect to definition length, while computing a digest initially requires $O(|d|)$ work over the encoded definition. Cached embeddings can also be compared without rereading definitions, but their similarity score answers a different, approximate question. Full-digest comparison tests exact canonical identity under the agreed scheme. A deployment that exchanges one verified context root and caches the result can amortize a one-round-trip handshake over that unchanged context; discovery, hydration, mismatch recovery, and context changes may require further work. Strict mode rejects a mismatch in the selected canonical context, while cooperative prefix mode provides a lighter drift-detection check. Placing an uncached exchange on a live high-frequency-trading execution path may be prohibitive; accepting that overhead at a safety-critical coordination boundary may be warranted. These are deployment trade-offs, distinct from the design-session timings reported below.

7.7 Demonstration: Multi-Agent Solution Design

This is an illustrative end-to-end demonstration: $N = 5$ runs per condition, one task domain, and one model family. It explores turn-count variability, the possible "Sema Tax" of vocabulary use, and the potential benefit of an optimistic coordination protocol. The deterministic identity mechanics and Pattern Card validation of Sections 3–4 remain the paper's primary contribution. The conditions bundle content-addressing with different prompts, workflow requirements, and the bootstrap library; isolating the protocol contribution would require matched instructions and an equally well-crafted non-hashed vocabulary, an ablation not performed here.

We ran three conditions with five multi-agent swarm runs each, tasked with designing a High-Frequency Trading (HFT) "Pump-and-Dump Detection Engine," chosen for its 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. Table 8 uses the archived harness summaries: turns are the reported `turns_executed`, elapsed time is `duration_seconds`, and range is the minimum–maximum turn count across the five runs. These measure design-session coordination, not deployed trading latency or design safety.[8] We compared:

- **Condition A (Natural Language)**: Agents coordinated without Sema tools or pattern references.
- **Condition B (Sema Workflow)**: Agents had Sema tools and used the `PUREBrainstorming` (`55d8`) and `PURECheck` (`0b12`) workflow with `MechanisticDesignProposal` (`ad31`) outputs, as reflected in the available configuration and archived messages. This condition lacked the prescribed OptimisticSolver/AtomicBid layer of C. Four of five archived runs invoked `sema_handshake`.
- **Condition C (Sema Workflow + Optimistic Protocol)**: The Sema workflow was coupled to the experiment-time `OptimisticSolver` (`ae61`) and `AtomicBid` (`4b82`) definitions, specifying that an agent bundle its intent declaration and tool execution in the same turn.

The patterns named in Conditions B and C are still in the library, in later versions (Appendix B); Appendix A gives the current OptimisticSolver card.

[8] The fifteen archived run summaries and event traces are retained under `experiments/sema_design_challenge/traces/`, alongside available YAML configurations. Those configurations are reproduction artifacts rather than immutable captures of the original prompts. For Condition A, the available configuration includes ordinary-language role, quality, and optimistic act-now instructions; the archived traces contain the corresponding intent-declaration syntax. The separate historical vocabulary snapshot, `experiments/paper_v1/taxonomy_v1.db`, retains the experiment-time definitions cited here, including OptimisticSolver (`ae61`), AtomicBid (`4b82`), and MechanisticDesignProposal (`ad31`); it is distinct from the current-library inputs used above.

Table 8: Multi-agent design-session results ($N = 5$ per condition)

Condition	Vocab	Optimistic Sema protocol	Avg Turns	Avg Time (s)	Turn range
A (Baseline)	×	×	22.6	317	4–51
B (Workflow)	✓	×	**14.8**	230	10–19
C (Optimistic)	✓	✓	15.4	**203**	8–25

The narrower observed turn-count ranges in Conditions B and C motivate the *Stability Hypothesis*: a shared, structured vocabulary may reduce coordination variability by giving agents common procedures and stopping criteria.

The Tail Risk of Natural Language (Condition A) The baseline showed the widest observed range, completing in as few as 4 turns but extending to 51 turns in a coordination loop that lasted over 13 minutes. The long run illustrates the practical cost of ethical debate and circular reasoning failing to converge. It motivates a tail-risk concern alongside research on consensus difficulty in natural-language multi-agent systems [33]; five observations do not establish a fat-tailed distribution.

The "Sema Tax" Hypothesis (Condition B) Condition B had the narrowest turn-count range (10–19 turns; sample standard deviation $s = 4.0$) and was slower on average than C (230 versus 203 seconds). A possible explanation is the coordination cost of looking up definitions and checking invariants—the "Sema Tax"—consistent with costs discussed in prior work [34, 35]. The runs did not separately time those activities.

The Protocol Multiplier Hypothesis (Condition C) Condition C had the fastest mean completion time. Its experiment-time `OptimisticSolver` (`ae61`) and `AtomicBid` (`4b82`) definitions bundle intent and execution, suggesting a way to offset coordination overhead by avoiding separate permission exchanges. OptimisticSolver's Tier 2 design (Section 4.3) assumes alignment-seeking agents and deliberately trades some verification overhead for lower latency. C had a wider turn-count range than B, but neither Sema condition reproduced A's longest loop in these five runs.

The traces also show the vocabulary being used to organize mechanism-oriented design and critique. In the first archived Condition C run, the team moved from static bytecode monitoring toward *Differential Trace Simulation*, then examined simulation-aware evasion and funding-lineage blind spots. This is a concrete example of agents proposing a causal mechanism, identifying how it can fail, and refining the proposal's stated constraints rather than merely citing pattern names.[9] The prescribed `MechanisticDesignProposal` (`ad31`) structure supplies a scaffold for explicit causal attack models. Its effect on design quality remains unassessed; a comparison would need matched output requirements and independent review.

8 Related Work

Sema combines several established lines of work. The comparisons below distinguish content identity, procedural vocabulary, interoperability and alignment, and runtime trust.

8.1 Content Identity and Communication

Secure Definition Hash (SDH) Hawke proposed content-addressing semantic definitions in 2002 [3], using identifiers such as `urn:sdh:<hash>:<name>`. SDH is a direct predecessor: a term binds its use to a hashed definition document, whose own terms can refer to earlier definitions. Sayers and Eshghi [36] independently proposed content-derived RDF URIs as shorthand and integrity proofs. Trusty URIs [4] show how hashing a canonical representation above raw serialization bytes lets identity survive reserialization. Sema builds on this definition-binding idea with a particular agent-facing realization: readable terms in natural-language messages refer to structured behavioral contracts, whose Merkle fields localize agreement and whose exact dependencies expose composition. The handshake adds a deployment policy for mismatch. Gustafson's content-addressed semantic-data work [13], combining RDF canonicalization with IPFS identifiers for Underlay, supplies a related semantic-data precedent.

[9] See `experiments/sema_design_challenge/traces/run_20251229_162004/C_Optimistic/swarm.jsonl`, especially the revision request at line 113 and later critique at line 146. These are illustrative design claims from the agents, not externally validated trading mechanisms.

Content Identity Across Domains InChIKey hashes a molecular InChI representation into a compact chemical identifier [37]; Git and IPFS [5] use content addresses for software and data. UniParc illustrates a related identity goal through a different mechanism: each unique protein sequence receives a stable archive identifier (UPI), rather than making that identifier a content hash [38]. These examples show why the representation and identity rule must be stated for each domain; they need not use the same encoding to motivate stable reference.

Ethereum ABI function selectors [39] dispatch calls using a 4-byte Keccak-256 hash of a type signature; Ricardian Contracts [40] identify a human-readable contract by its hash; and Dhall [41] hashes normalized expressions rather than source bytes. These precedents respectively connect identifiers to dispatch, contractual text, and canonical form.

Agent-Communication Precedents The Agora Protocol [15] is a close agent-communication precedent. Its protocol documents can be identified by a SHA-1 hash, while messages without a selected protocol remain possible [42]. Its reported 100-agent demonstration exhibited emergent protocols without human intervention. Agora's document-level identity and Sema's per-field behavioral-contract commitments address different comparison granularities: the latter can locate a changed invariant or dependency without treating the whole contract as an undifferentiated mismatch. Spritely Goblins' Content Addressed Descriptors [43] provide another communication precedent: a structured-description hash becomes a method name in OCapN actor-model communication.

Code, Cognition, and Economy Pattern Cards place reasoning procedures such as `Reason#81ec` and economic acts such as `Bid#647f` in the same graph. The intuition is that an agent can inspect its commitments with the same compositional grammar it uses for its reasoning. Unison [6] identifies functions through hashes of de-named abstract syntax trees. Pattern Cards apply a related definition-DAG idea to behavioral specifications, adding typed interfaces and handshake comparison. The name/identity separation has a precise limit here: renaming an object's own unhashed handle preserves its digest, but handles retained inside hashed references can affect dependent identities.

8.2 Procedural Vocabularies and Reasoning

Pattern Languages Alexander et al. [44] introduced pattern languages for architecture; Gamma et al. [45] adapted them for software. These provide taxonomic structure but not content-addressing or machine verification. Pattern Cards combine pattern-language structure with cryptographic identity.

Retrieval-Augmented Generation (RAG) While traditional RAG systems retrieve unstructured text chunks to ground LLM generation [46, 47], the current tooling retrieves structured, content-addressed procedural definitions. Instead of grounding a prompt only with static knowledge, it can hydrate named procedures whose canonical identity the runtime verifies.

Recursive Language Models Zhang et al. [48] place a long prompt in an external environment where a model can inspect it programmatically and delegate selected passages to recursive model calls. If several calls apply the same rule to different passages, a Sema reference could let each verify which definition of the rule it received.

Portable Capability Definitions Voyager [49] stores executable skills that an agent learns in Minecraft and retrieves them for later tasks. Pattern Cards describe procedures that agents can exchange by reference and verify down to their declared dependencies. Agent Skills [32] package a named Markdown instruction body with metadata and optional scripts, references, and assets. A skill's declared string name does not itself verify the body's content; a Pattern Card's full identifier binds its canonical contract. Skills can reference resources and invoke code, while their base format does not require the exact content-addressed dependency graph used by Pattern Cards. A2A Agent Cards [50] and MCP tool schemas [51] likewise describe discoverable capabilities and structured interfaces.

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 [8] introduced typed signatures for language model pipelines, and DSPy Assertions [9] added computational constraints (`dspy.Assert`, `dspy.Suggest`) that are structurally similar to Pattern Card contracts. White et al. [10] provided the foundational work on formalizing prompt patterns using software design pattern structure. Leoveanu-Condrei [52] 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: hashing a contract into its identifier makes identity of the resolved canonical contract verifiable by digest comparison; it does not establish that differently worded contracts are semantically equivalent or that a runtime satisfies the contract.

Agent Coordination Frameworks Frameworks such as LangChain, AutoGPT, CrewAI, and OpenClaw [53] organize agent roles, tools, and interactions. Sema integrates through its MCP server, allowing those workflows to resolve and compare the behavioral contracts they use.

The companion work on thinking protocols [21] asks how to structure not only *what agents do*, but also *how reasoning is divided*. Pattern Cards make the cognitive handoff explicit: before a creative solver delegates to a verification solver, a compliant handshake can verify that both resolved the same contract for a valid result. This is the contract-verification substrate used by the companion architecture.

Failure Analysis and Verification The MAST study [34] analyzes failures across more than 1,600 execution traces from seven multi-agent frameworks and emphasizes specification and coordination failures. Those categories motivate explicit contracts, but do not identify the fraction caused specifically by mismatching external definitions. AgentSpec [54] addresses runtime rule enforcement, complementing Sema's definition-identity layer.

8.3 Interoperability and Alignment

The 2025–2026 Protocol Landscape The cited interoperability landscape [55] distinguishes complementary functions: **MCP** [51] connects agents to tools and data; **A2A** [50] supports task delegation through capability-based Agent Cards; **ACP** provides asynchronous messaging; and **ANP** [56] uses decentralized identifiers and JSON-LD for discovery. Their different capability labels—Tools, Skills, Operations, and shared vocabularies—illustrate why transport agreement alone does not establish identity of a behavioral definition. Sema references can travel within these existing transports.

Emerging Semantic Alignment Approaches Symplex [57] proposes semantic-intent vectors and cosine-similarity negotiation: a fuzzy match can tolerate paraphrase, but supplies no exact-content identity guarantee. NLIP (Ecma-430 through Ecma-434) [58] uses AI-mediated translation between local ontologies. These approaches ask whether representations can be aligned or used together; Sema's strict handshake asks whether selected external definitions are identical under an agreed scheme. Similarity, translation, and exact identity can therefore serve different stages of coordination rather than substitute for one another. Table 9 compares their targets and assurances. Berdoz et al. [33] ask a prior question, whether LLM agents can reach agreement on a value at all, and find that they often fail to; that concerns agreement on decisions, not on definitions.

Table 9: What each approach checks. Exact content identity, provenance, and acceptance policy are different assurances; Sema's halt policy requires an enforced handshake.

Representative	Comparison target	Assurance or decision basis
Symplex [57]	Intent embeddings	Similarity under a vector metric
Fleming et al. [59]	Signed definitions	Issuer authentication
NLIP [58]	Local ontologies	Model-mediated translation
AOW [60]	Enterprise vocabulary use	Governance and review
Agora [15]	Protocol document	Document-content identity
BlockA2A [16]	Interaction records	Message provenance and integrity
Git, IPFS [5]	Addressed objects	Object-content identity
Sema	**Referenced definitions**	**Canonical-content identity; enforced mismatch halt**

Agentic Ontology of Work Skan AI's Agentic Ontology of Work (AOW) [60] supplies an enterprise vocabulary for describing and governing intelligent automation, including Objectives, Intents, Agents, Skills, Policies, Outcomes, Assurance Levels, Memory, and Guardians in JSON-LD and RDF/OWL. Its taxonomy and governance address what terms should mean and how teams use them; Pattern Card verification checks which exact definition participants selected. Governance and content identity address complementary responsibilities.

Classical Vocabulary Alignment Chocron and Schorlemmer [35] developed formal interaction-based alignment techniques for heterogeneous agent vocabularies. Such alignment reconciles different vocabularies; Sema's strict comparison starts from a shared content-addressed definition. Reconciling differently worded equivalents remains outside that comparison.

Emerging Agent Infrastructure The W3C WebAgents Community Group's interoperability report [61] connects classical multi-agent research, including FIPA and BDI, with LLM-based protocols. Josifoski et al. [62] frame agents as modular semantic processors exchanging typed messages, while NLWeb [63] uses Schema.org

descriptions and MCP access to expose web semantics to agents. Each concerns how agents exchange or expose meaning; Sema references can travel inside such messages and descriptions, letting participants check that they resolved the same definition.

8.4 Authentication, Governance, and Enforcement

Identity and Trust Models Verifiable Credentials [64] authenticate issuer-backed claims; a content hash identifies the definition being endorsed. A participant may need both checks. A service such as Linked Open Vocabularies [65] supports discovery, while the content-identity check remains independent of who hosts the vocabulary.

Protocol-Layer Approaches Fleming et al. [59] propose an "Internet of Agents" with a Semantic Negotiation Layer (L9) and federated Schema Authorities for shared context. Sema's Pattern Card application emphasizes definitions of reasoning and coordination procedures—such as `BayesUpdate#8656`, `SteelmanCheck#b004`, and `LatticeCommit#488a`—that agents can cite inside natural-language messages. The distinction concerns scope, placement, and authority: a vocabulary may use institutional curation while its definition-equality check depends only on verified content under an agreed scheme.

BlockA2A [16] anchors interaction records through decentralized identifiers and a blockchain ledger. It concerns the provenance and integrity of messages; Sema verifies the external definitions those messages reference.

Execution Authorization and Message Admission Faramesh [66] places an Action Authorization Boundary between proposed actions and execution, with canonical action records, deterministic `PERMIT/DEFER/DENY` decisions, and fail-closed defaults. CLBC [67] gates messages through a verifier against a pinned predicate to bound covert coordination. Their responsibilities complement a Pattern Card handshake: definition identity concerns the contract named, message admission concerns what may enter the transcript, and execution authorization concerns which action may run. Together they suggest a three-layer safety architecture, whose integration remains future work.

8.5 Conceptual Lineage and the Training Connection

The Leibnizian Dream Sema's definition-coordination application 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 [68]. 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. Unlike Leibniz's top-down prescription of a perfect language, the vocabulary can evolve bottom-up as participants mint content-addressed definitions.

The Living Taxonomy Earlier work [7] sketches a self-authoring semantic infrastructure with two complementary roles. At runtime, named patterns embedded in natural language let agents invoke reasoning without re-explaining it, illustrated by entries such as "Pattern #7,832: Statistical Intimidation." In training, the proposed objective shifts from $P(\text{text}|\text{context})$ to $P(\text{text}, \text{thinking}|\text{context})$, with a metacognitive corpus citing the same patterns. The work also warns of "Cognitive Architecture Wars" under centralized control and proposes on-chain governance as one possible response. Sema supplies a concrete runtime substrate through hash-based references, such as `BayesUpdate#8656`, whose identities can be verified independently of an institution.

The proposed connection is a feedback loop: runtime use produces explicit pattern references and candidate refinements; a metacognitive training corpus can cite those artifacts; and training or evaluation can inform which formulations prove useful in later coordination. Shared content identities make the objects of that loop inspectable across stages. Testing this feedback loop remains open work.

8.6 Contribution Summary

Against this landscape, Sema makes seven contributions, separable into the representation-agnostic communication primitive and its JSON/Merkle realization (C1), the Pattern Card agent-coordination layer (C2–C6), and bootstrap-library design (C7).

C1: Hash-as-word inversion. Sema moves content-addressed identity into text-carried references that can function as words, independent of the representation being hashed. These references are the hash-based verifiable words of

the abstract. This paper realizes the primitive with canonical JSON/Merkle hashing. In the Pattern Card layer, a reference such as `Delegate#00fa` combines a readable handle with a short locator that must resolve to, and be verified against, the full identifier for collision-resistant assurance.

C2: Hashable behavioral contracts. In the Pattern Card layer, the hashed object is a behavioral contract: mechanism, invariants, preconditions, postconditions, typed dependencies, and failure modes. Full-digest verification establishes identity of canonical form, not semantic equivalence; two behaviorally equivalent contracts with different wording still produce different hashes.

C3: Field-level comparison. Pattern Cards are hashed field-by-field and rolled up through a Merkle tree. Field hashes allow agents or tools to compare selected fields independently; coordinating on a matching subset requires a separate policy, while the current handshake remains fail-closed on root mismatch.

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 collision-resistant coordination requires equality of full identifiers or full roots. In strict mode, `sema_handshake()` proceeds only on a full match and halts on mismatch; cooperative prefix mode provides drift detection rather than full assurance. The guarantee applies to compliant deployments that invoke and enforce 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 617-pattern vocabulary organized by dependency layer and by agency role: actions, data shapes, and coordination/evaluation primitives. This is a second-order contribution: the communication primitive is representation-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. They are given in four groups: two on what a matching hash cannot establish; four on what the implementation and the library's rules do not yet settle; four on what the evidence does not yet show; and two on what remains open for a community.

The Synonymy Limit. Behaviorally equivalent cards with different invariant phrasings or decompositions can have different canonical content and therefore different hashes. The Taxonomist workflow and embedding audits support the curation needed to recognize such synonyms. Content addressing allows a community fork with equivalent formulations to coexist with the canonical library; merging them requires explicit alignment work outside the cryptographic layer.

Definition Quality. Weak invariants, incomplete failure modes, or an unsound mechanism can propagate bad coordination through perfectly valid references. Adversarial contract review can expose specification errors, but behavioral quality depends on authorship, review, and empirical testing. The protocol provides no defence against a well-hashed bad idea.

Historical Version Storage. A full Sema ID remains a content identity, but it can be resolved and verified only while a registry retains the corresponding content. The current GraphStore indexes one active definition per handle and does not archive replaced versions. Direct minting therefore rejects a parent replacement with inbound exact pins, while `sema apply` and pull validate the complete projected corpus before permitting a reviewed batch. Keeping old pins resolvable requires a future multi-version content store. The unhashed `_meta.supersedes` field records replacement claims but is not itself an archive or a traversal implementation.

Mandatory Coordination Boundaries. Making the handshake compulsory requires integration into the agent's execution harness: coordination must be gated on successful verification, rather than leaving tool invocation to the agent. The compliant-deployment guarantee in Section 3.5 depends on that integration, which lies beyond the current tool interface.

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 4.2) narrows the judgment but does not eliminate it.

From Specification to Operation. Realizing the proposed Genesis Loop requires a scoring implementation for `Judge#704c` and integration of candidate generation, review, and minting into an automated process. Contextual Telepathy needs snapshot restoration, while Experiential Telepathy needs compatible learned-state serialization and import (Section 5.4). These mechanisms need experimental validation beyond the implementation status summarized in Section 6.

Demonstration Scale and Coordination Cost. Beyond the five-run demonstration, a larger evaluation should vary architectures and domains, match prompts and non-hashed vocabulary content, and define outcome scoring in advance. Separately timing vocabulary lookup, invariant checking, and coordination would test the proposed "Sema Tax" across model families and context-window sizes. These comparisons would distinguish vocabulary, tooling, and protocol effects.

Single-Agent Reasoning Scaffold. The vocabulary's Mind layer may serve as an internal scaffold: resolved reasoning patterns bring explicit failure modes and invariants into an individual agent's context. The present demonstration exercises coordination, so this single-agent use remains unevaluated. The theoretical foundations for composing reasoning across solvers are developed in [21].

Empirical Pattern Validation. Pattern-level usefulness also needs testing beyond the analytical review in Section 4. Relevant comparisons include tasks using `StateLock#c9c2` versus ad-hoc prose, Mind-layer patterns versus unstructured chain-of-thought, and alternative formulations of the same mechanism. A/B comparisons on realistic tasks could feed observed reliability, solution quality, and clarity into refinement (Section 4.8), selecting for practical utility as well as internal coherence.

Embedding Granularity. Larger or domain-specific encoders may reveal clusters missed by the current similarity screen. The 25 pairs at or above 0.70 warrant documented comparison of meaningful variants and possible synonyms.

Vocabulary Governance. Permissionless minting creates governance challenges at scale: namespace pollution, adversarial pattern injection, and quality degradation without curation. How well selection through the proposed `Judge #704c` would resist those pressures remains unevaluated.

Alignment Positioning. A vocabulary of *beneficial* strategies—conflict de-escalation, informed consent, non-coercive persuasion, care under uncertainty, adversarial input handling—could use the same mechanisms as the coordination library: verified definition identity, inspectable typed dependencies, and an enforced mismatch boundary. Encoding such strategies as Pattern Cards is a direct application of Sema; determining whether they are beneficial across relevant contexts requires substantive curation and evaluation. A vetted library could make those strategies easier to invoke precisely at runtime.

Precision alone does not determine value alignment. The same identity mechanism can support beneficial or harmful coordination, depending on the definitions, agents, and surrounding constraints. A Sema-equipped system therefore still needs aligned agents and appropriate governance and enforcement. The complementary metacognitive training proposal in [7] seeks to weave beneficial values into reasoning through $P(\text{text}, \text{thinking}|\text{context})$ training. That proposal addresses how agents learn and use their strategies; a beneficial-strategy vocabulary supplies explicit, reviewable contracts for runtime use.

10 Conclusion: The Substrate, Not the Law

Sema is a minimal communication primitive: hash information under an agreed representation and carry its content address in communication. This paper realizes the primitive through canonical JSON/Merkle hashing, instantiates it as Pattern Cards, and supplies a 617-card bootstrap library.

In the Pattern Card application, hashing a definition anchors a reusable word. Changes to hashed content reveal divergence, so an enforced handshake can catch mismatch before coordination. The result is compact reference backed by verification, precise definition coordination, and permissionless vocabulary growth.

Building on content-addressed systems such as Git, IPFS, Unison, and SDH, Sema makes readable, hash-anchored definitions available as terms in natural-language agent messages: the unit of verification enters the unit of communication. The analogy with *movable type* captures the intended change in use. Standardized, reusable units become native to a medium; here, participants can mint, resolve, and recombine words backed by exact definitions. Any text channel can carry them. Whether reusable definitions produce compounding network benefits as the vocabulary grows is a hypothesis for the resulting commons.

The initial vocabulary is a seed rather than a law: 617 patterns form one concrete starting point from which agents can coordinate and mint alternatives. Refining it treated ontology maintenance as architectural engineering. Within this library, classifying patterns by structural execution requirements proved a useful guide to resolving tensions that subject-matter categories had obscured—the intuition behind "topology trumps topic." Whether that lesson generalizes remains empirical.

Sema holds no authority over what information may be addressed or how vocabularies are organized. Users can mint alternatives, preserve competing interpretations, and let namespaces evolve. The larger aim is a self-governing commons of thought: a substrate through which agents can inherit and improve explicit conceptual distinctions, rather than a requirement that this particular vocabulary endure.

Under the agreed scheme: same verified digest, same hashed content.
For a selected contract in strict coordination: mismatch, halt, and clarify.

Code and Artifact Availability

Code, paper source, vocabulary data, and release artifacts are available at `https://github.com/emergent-wisdom/sema`. The version series is archived at doi:10.5281/zenodo.19462702.

A Pattern Card Specifications

This appendix contains automatically generated current specifications for six selected examples: StateLock, FractalIntelligence, SpectralTune, SteelmanCheck, WhyClimb, and OptimisticSolver. They illustrate the vocabulary discussed in the main text; the historical multi-agent demonstration uses the experiment-time definitions identified in Section 7.7.

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:c9c2809bdf97bb6ac17a8ffe428d4aa9a28294b99d64cc8a336a239d26a8afb8

```
{
  "dependencies": {
    "composes_with": {
      "backoff": "sema:Backoff#mh:SHA-256:9
          e59718422f5408d7892feccb6d538a21103f9573b122b891ee3c8cf66303e72",
      "cooldown": "sema:Cooldown#mh:SHA-256:
          c288d88055cb0aae80c4b67ed62afcd3ebcc5de18749fc6efb15e9f8c529b9df"
    },
    "references": {
      "actor": "sema:Actor#mh:SHA-256:86
          fc0930b97ffb4fe85d4820af8e17a43166b745d181895f576edf39edde6aa8",
      "lock": "sema:Lock#mh:SHA-256:95
          c2ee952a5301d4b346a5e8693350829d88b3c600048eedcf87bb00c23ee5fb",
      "state": "sema:State#mh:SHA-256:4
          f25e6a16ccd37dc0bc154cccc7055cb504d49daa8b712f372a425eb6c7b8cb2"
    }
  },
  "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",
  "invariants": [
    "Both {{actor}}s sign every write to the fused {{state}} subset.",
    "Exclusive over that subset: no third party writes it while the {{lock}} holds.",
    "Temporary: the fusion ends on completion or timeout, never implicitly."
  ],
  "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-256:77398a38a5c907b5e94cc5ec7919f6a198299f3397c5d580783da9cd1e8b3d77

```
{
  "dependencies": {
    "composes_with": {
      "localized_learning": "sema:LocalizedLearning#mh:SHA-256:a6dbe910a57206b1a10d3b60733dfe3b98fa85696e701674abe860ee0b730950",
      "marginal_value_rule": "sema:MarginalValueRule#mh:SHA-256:661109205c507b2c26d683d091de7f8b782bf2d6c2e9192373b242a9b078ac71",
      "polymorphic_solver": "sema:PolymorphicSolver#mh:SHA-256:ce9a27d50265e6e74e034eb4da3a6441d840b75a740772da3d577d9602727f1b",
      "problem_framer": "sema:ProblemFramer#mh:SHA-256:ad6b7fc1f14f2e26dd0d230f81382a2c602c365ebe4b9ad5fd390fb0c2d4c4fc",
      "reason": "sema:Reason#mh:SHA-256:81ec8196c7c1c761fe353c8d26463bcb61b54f844aa3422a40019f82ef202321",
      "recursion_dive": "sema:RecursionDive#mh:SHA-256:a86f17bad949a7a5405f04b4bc43a3e6be28c5c31e8328ccf178fa292719d754",
      "reframe": "sema:Reframe#mh:SHA-256:cbfd4fadde729367ef8f32a68951e5852535b7f7c3edc68a502d9187b8c1d54a",
      "solver_decomposition": "sema:SolverDecomposition#mh:SHA-256:ff4b4921d148c7672540eb0ec6201e5eed9b7c06b6442c6525399a856f121bfa",
      "state_snapshot": "sema:StateSnapshot#mh:SHA-256:d75752f332fe14ae3abf2253a45ab180034d2becc3a7e19e3e30da3c94b3c636",
      "synthesis": "sema:Synthesis#mh:SHA-256:4f1fecba4183e6a79bfb3bd8f1981a6b6fb9f3dced92a484a9aa466c93ae249b"
    },
    "references": {
      "agent": "sema:Agent#mh:SHA-256:e127459069739fffc96a71d54ef6e8c97ca53471d5349c16aa72a092a33756cd",
      "conservation": "sema:Conservation#mh:SHA-256:0b32d007b27a9b5201c46931bdbabd09d08ff9d7b01cf49469f61709989507c2",
      "experience_sharding": "sema:ExperienceSharding#mh:SHA-256:50ff836c2fc73c18fed683db6e3707a18525da22b100f3db23b6fd7604a8a85e",
      "specialize": "sema:Specialize#mh:SHA-256:799d587f9e409c7f9fbf13f5dc83a2437068df8a0738f9bcfb910ee00c0c20a1",
      "strategy": "sema:Strategy#mh:SHA-256:3dc598d0c6af12389df445c204e17df03950dbc886656b85700e6c87c31b2eef",
      "system": "sema:System#mh:SHA-256:f8eb318a1113a10fead02fbf14ae433767ac1a69c9579cdb584cc622068e90b3",
      "task": "sema:Task#mh:SHA-256:a85010f508f85055170c396c8fa390125c1206888c565ac10651f3969a339f8d",
      "universal_solver_tree": "sema:UniversalSolverTree#mh:SHA-256:f9fba480ef013c98691c0ccb6cbbd58dab5183e60798c550ff0de093dbd30727"
    }
  },
  "signature": [
    "System(Reason)"
```

```
  ],
  "mechanism": "Expansion of cognitive capability through {{solver_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:c0aab008d2cd2f653e56508356987477f9207b4cf6435f4416105b3f37bd544a

```
{
  "dependencies": {
    "accepts": {
      "signal": "sema:Signal#mh:SHA-256:6
          eaff8a06e08e75279670bc09d88175d130815240fdfb3a21252c0364769c1fc"
    }
  },
  "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": [
    "No payload data is processed before Tune_ACK.",
    "On hash mismatch, halt and escalate rather than retry."
  ],
  "preconditions": [
    "Both agents possess valid hash function H"
  ],
  "postconditions": [
    "Channel is established with matching context hashes, OR connection terminated
        and escalated"
  ],
  "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"
    }
  ],
  "failure_modes": [
    "Over-strict halt: a benign version skew or encoding difference produces a
       hash mismatch and halts a channel whose parties would have understood each
       other. Hash equality admits no partial match, so any difference is total."
  ]
}
```

A.4 sema:SteelmanCheck#mh:SHA-256:b004eb6d85a184da7f9bb7fc06d011bc869f6eb6ad1061f30cbbbcc4c2b8f6aa

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

```
{
  "dependencies": {
    "composes_with": {
      "check": "sema:Check#mh:SHA-256:6890
          b221c38531710528cd84bbf3a957e29d9836dfc9c02669fc0a86709bcb00",
      "critique": "sema:Critique#mh:SHA-256:646
          b71acb10a6b7f0874f3d5c7977c44d253f29f9c1231c974d85c7fba131980",
      "judge": "sema:Judge#mh:SHA-256:704
          cbb71588d7a3bafa08b71c863005e31ec4400860b9cb8b094e660da9c7c6c"
    },
    "references": {
      "agent": "sema:Agent#mh:SHA-256:
          e127459069739fffc96a71d54ef6e8c97ca53471d5349c16aa72a092a33756cd",
      "belief": "sema:Belief#mh:SHA-256:09
          e1c919d9829e30422f3fffee11c1af4af54ac7894d157ff371ed50b03f69b1",
      "cognitive_bias": "sema:CognitiveBias#mh:SHA-256:
          c8a4abe2e0ce048be16d25f890a50f8c7095e099b5ab2de23aa059942328f387",
      "decision": "sema:Decision#mh:SHA-256:
          b1407127acd4a39314fc24b418310e4b95e0347e4e22f376151ab765e2dfd2c7",
      "loop": "sema:Loop#mh:SHA-256:37
          f953c0b21b9b0c72a639c4e1a54354faa71f07905531d271f6bbf40c341603",
      "robustness": "sema:Robustness#mh:SHA-256:5304
          c7824b76c6f308bb1a33ba6171d41fd061a5b104f21710bbb6c7a7c85b43"
    }
  },
  "signature": [
    "Check(Robustness)",
    "Critique(Belief)",
    "Judge(Critique)"
  ],
  "mechanism": "Before finalizing a {{decision}} or output, the {{agent}} MUST
      generate a claim-specific counter-argument to its conclusion. It uses {{
      critique}} to challenge the underlying {{belief}}, {{judge}} to score the
      counter-argument, and {{check}} to test the {{decision}}'s {{robustness}}
      against any counter-argument that meets 'strength_threshold'. A sub-threshold
       counter-argument is strengthened up to 'iteration_limit'; if no adequate
      counter-argument is produced, release is blocked. Once an adequate counter-
      argument exists, a Falsified robustness result requires revision or discard,
      Unknown blocks release pending review, and only Verified permits finalization
      . It prevents confirmation {{cognitive_bias}}. For adversarial contexts, see
      adversarial steelmanning.",
  "gloss": "Qualify the opposition, then release only if the conclusion remains
      robust",
  "invariants": [
```

```
    "Adequate opposition: before release, {{judge}} scores the counter-argument at or above ‘strength_threshold‘.",
    "The counter-argument identifies the specific claim it contradicts.",
    "Release discipline: only a Verified {{check}} of the {{decision}}’s {{robustness}} after adequate opposition permits release; Falsified requires revision or discard, and Unknown requires review."
  ],
  "preconditions": [
    "A {{decision}} or output exists to challenge.",
    "The conclusion exposes a specific claim or rationale that can be contradicted."
  ],
  "postconditions": [
    "Either bounded failure to produce an adequate counter-argument blocks release, or an adequate counter-argument and terminal {{check}} result are recorded; the {{decision}} is finalized only on Verified."
  ],
  "parameters": [
    {
      "name": "iteration_limit",
      "type": "Integer",
      "range": "[1, 5]",
      "description": "Maximum strengthening attempts before failure to produce adequate opposition blocks release"
    },
    {
      "name": "strength_threshold",
      "type": "Float",
      "range": "[0.7, 0.95]",
      "description": "Minimum {{judge}} score for a counter-argument to qualify as a steelman; it is not the verdict on the {{decision}}"
    }
  ],
  "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:WhyClimb#mh:SHA-256:ed20580f4eecde83bd4822747358418d4b4265c40e468f80b870cd63d3817c93

```
{
  "dependencies": {
    "references": {
      "condition": "sema:Condition#mh:SHA-256:b480f3c4da11ba6d86f6d14e0fb8521268f3c4bb38588460f7ff4253ad8d78d5",
      "problem": "sema:Problem#mh:SHA-256:2580c43677ec78ee7ee428d200662c6096e4eab38c2c5a868154979074cc0f08",
      "recursive_root_cause": "sema:RecursiveRootCause#mh:SHA-256:f132a6796ffbbddea29f4fe56ff3f81442c99390999348154b6f4ee146be4ae6",
      "reframe": "sema:Reframe#mh:SHA-256:cbfd4fadde729367ef8f32a68951e5852535b7f7c3edc68a502d9187b8c1d54a",
      "solution": "sema:Solution#mh:SHA-256:8d55004ffa8f61d6e7d12d588ae4baa9723296dd7fa501759b5e6b1b97e7c9ab"
    }
  },
  "signature": [
    "Reframe(Problem)"
  ],
```

```
  "mechanism": "A root-cause analysis protocol where the agent iteratively asks '
      Why is this a problem?' to {{reframe}} the {{problem}} and ascend the
      abstraction hierarchy. The goal is to reach the 'Ceiling'\u2014the highest
      level where the problem is still actionable but the solution space is
      maximized. Utilizes {{recursive_root_cause}}.",
  "gloss": "Recursive problem abstraction",
  "invariants": [
    "Ascension: Scope(Level N+1) > Scope(Level N)",
    "Stop {{condition}}: Halt when 'Why' leads to a value judgment or physics
        constraint"
  ],
  "preconditions": [
    "{{problem}} is well-formed"
  ],
  "postconditions": [
    "{{solution}} space expanded"
  ],
  "failure_modes": [
    "Climbing too high (solving 'Entropy' instead of 'Fix Bug')."
  ]
}
```

A.6 sema:OptimisticSolver#mh:SHA-256:1e7272dcb783bde00719114714447261d80110e0f9983372e1453cbbe6d6ef1c

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

```
{
  "dependencies": {
    "composes_with": {
      "atomic_bid": "sema:AtomicBid#mh:SHA-256:5
          ac637b4b3d74bfaea558ee36b1ee6ffcaf46fe26ac988b524e46faf9821535e",
      "compensate": "sema:Compensate#mh:SHA-256:4
          b87623f38564fcc95b87d2cbf5df2f5ff41ef27e617ccafd76c7b7896b144e7",
      "compute_budget": "sema:ComputeBudget#mh:SHA-256:4091
          ac7c6677342920dc53808d02830e35c5849f2af1bb9ba7749c576d37478e",
      "pathway_memory": "sema:PathwayMemory#mh:SHA-256:
          ce2e6ce822404fae0e60a43fc2c6fbe2e4d51ab9aefee1e7a9e21da75893e722",
      "reflexion": "sema:Reflexion#mh:SHA-256:81
          eb9140703ea95f53556741df731af84c1cada66cc1e0f3a1d88e4de5fe54d8"
    },
    "references": {
      "parallel": "sema:Parallel#mh:SHA-256:
          d58d6696be11be09651e71307f0ea6f42bb45fac137ed985a61e475144f687a5",
      "polymorphic_solver": "sema:PolymorphicSolver#mh:SHA-256:
          ce9a27d50265e6e74e034eb4da3a6441d840b75a740772da3d577d9602727f1b",
      "rigorous_solver": "sema:RigorousSolver#mh:SHA-256:
          d3e3625a3e5a7fc0da9ec67377e8e60c0dd76d730afd053af02d5833e947ea39"
    }
  },
  "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."
  ],
  "extends": "sema:PolymorphicSolver#mh:SHA-256:
      ce9a27d50265e6e74e034eb4da3a6441d840b75a740772da3d577d9602727f1b"
}
```

B Category Taxonomy

This appendix presents the complete taxonomic hierarchy of the bootstrap library.

The 617 patterns are organized into 14 category names across 18 layer-category paths and 4 fundamental layers:

Physics Layer (22 patterns) The immutable substrate.

- **Dynamics** (6): Brakefade, Ripplelag, Swayhold, Syncdrift, VirginSoilEpidemic, WaveOfAdvance.
- **Primitives** (15): Attractor, Causation, Conservation, Dampen, Decay, Distance, Entropy, Equilibrium, Gradient, Lock, Measurement, MutualInformation, Noise, PhaseTransition, Reversibility.
- **Time** (1): CausalBarrier.

Infrastructure Layer (173 patterns) Operational constraints.

- **Data Structures** (101): AcceptSpec, Aesthetics, Anomaly, Artifact, Assessment, Assumption, Audit, Ballot, Belief, Boolean, Break, Cache, Card, Category, Chain, CognitiveBias, CompositionEdge, ConceptAnchor, Condition, Constraint, Context, Contract, Correlation, Criteria, Cyclic, DAG, Datum, Decision, Event, Exception, ExecutionManifest, FailureTrace, Forest, FrameSpec, Goal, Hierarchy, Hypothesis, Identity, Jointscape, Ledger, MECE, MechanisticDesignProposal, Message, Meta, Metric, MetricReading, Mode, Morphproof, Nature, Option, Outcome, Overlap, Parallel, PerformanceSignal, Permission, Plan, Probability, Problem, ProblemSpace, Prompt, Proposal, ProtoPack, Protocol, Prototype, Queue, Resource, Result, Risk, Role, RolloutManifest, RuleSet, Score, ScoringFunction, Sequence, Shard, Signal, Skeleton, Snapshot, Solution, SolverManifest, Spec, SpecializationEdge, State, Status, Step, Stream, StyleSpec, Subject, Summary, Sunsetkind, System, Task, Tension, Topology, Trait, Transition, Tree, Value, Variable, Vector, Work.
- **Dynamics** (3): Formelt, Limitblur, Regulapse.
- **Primitives** (54): Act, Actor, Aggregate, Backoff, Branch, Budget, Care, Check, CircuitBreaker, Combine, Compare, Compensate, Compress, Cooldown, EntropyPump, Equivalence, ExponentialBackoff, FailClosed, Feedback, FeedbackSignal, FinalizedIntervalRead, Gate, Greet, Heartbeat, Hysteresis, IdempotentWrite, Incongruity, Judge, Loop, Monitor, MonitorReport, Mutex, NegativeProof, Observe, ParetoFront, Probe, Quorum, Rank, ReAttempt, Route, Sandbox, Search, Select, Sign, StateAudit, StateSnapshot, StateTransition, TaskLifecycle, Throttle, TimeWarpLog, ToolInvoke, Trace, TriGate, Warmup.
- **Safety** (1): ExpandContractMigration.
- **Verification** (14): AuditTrail, ClosureConservationCheck, CompatibilityCheck, DetectorSelfTest, ExplainBeacon, FailurePreservingReduction, HumanApprove, InputGuard, MetamorphicTesting, OathBind, OutputGuard, ReleaseArtifactVerification, SpotAudit, Validate.

Mind Layer (266 patterns) Mechanisms requiring cognition; solver contracts support decomposition and delegation.

- **Dynamics** (14): Aimscatter, Capacitylag, Crossripening, Echohollow, Echoverturn, Foundationfray, Futurefold, Identitylag, Labelloop, Meaningthin, Mirrorbloom, Quietroot, Readyrise, RegressionToMean.

- **Inference** (27): AbsenceEvidence, BaseRateInclude, BayesUpdate, BreadthGovernor, ConfidenceCalibrate, ConfirmationBlock, ConfoundedAttribution, ContextFirst, EpistemicCalibrate, EvidenceDependence, EvidenceUnderdetermination, HackDetect, HindsightBlock, LayeredCheck, NormCheck, NormativeJudge, OntologyAdapt, Prestigeloan, ProphetFanOut, RegimeSense, ScopeFreeze, SemanticTabu, SourceEvaluate, SurprisalUpdate, SurvivorCorrect, TemporalEnsembleForecasting, TruthseekingProtocol.

- **Memory** (18): ArchiveServingFrontier, BeliefTracking, ChunkMerge, ContextCompress, CurriculumReplay, ExperienceSharding, HolographicShard, LatentAttachment, LocalizedLearning, PathwayMemory, Proprioception, Reasonfade, RetrievalAugment, Revisance, Scratchpad, SelfReminder, SimulationTrace, TraceBelief.

- **Reasoning** (102): Abduction, Almostnot, Analogysnap, Askwiden, Axiom, BackwardChain, Badgeread, Bisect, Blindmark, Callshort, Canonseal, ChainOfThought, CiteBack, ClassificationReview, CognitiveEcho, CollaborativeWritingProtocol, Complexitytax, ConceptualDecomposition, ConstructOntology, ContrastClass, Critique, Critiquedrift, Crossborn, Decompose, DecompositionGate, Deduction, DeepResearch, Dialectic, Edgehold, EffectDelayCheck, Eliminate, EpistemicCascade, Estimate, EthicalReasoningProtocol, Expansive, ExtendedThinking, Farsimple, Fermi, FirstPrinciples, FrameError, Gainmark, Generalize, GraphOfThought, HeuristicSnap, HumanEmulatorProtocol, Induction, Interpret, Invert, Kinproxy, Labelsqueeze, LeastToMost, Lensweave, LivedProof, Lossprint, Lucence, MandatoryAbstraction, MeaningChangeSweep, MetaPrompt, Motivejump, Nestmap, Normance, Parsimony, PatternDiscovery, PatternEmergence, QuestionSubstitution, ReAct, Realizable, Reason, RecursionDive, RecursiveRootCause, Refine, Reflexion, Reframe, RequestFraming, Roundtrace, RoutingSanityCheck, SelfConsistency, Situfit, SkeletonOfThought, SocraticLoop, SolverDecomposition, SpecBinding, SpecIntentCheck, Specialize, Spiralback, Statuslens, SteelmanCheck, StepBack, StrategicReading, Summarize, Synthesis, Termdrift, Think, Translate, TreeOfThoughts, Twinread, Twoledger, Uncertain, Understand, Verification, Webtrace, WhyClimb.

- **Strategy** (105): ActionConvergence, AdversarialSteel, Agent, AnalogyBridge, AntifragileInversion, BeamSearch, BoundaryReview, Brakegrow, Bubble, Build, CapacityPressure, CommitmentDevice, Compose, ComputeBudget, ConceptBlend, ConstraintFirst, ContingencyPlan, Creative, CreativeBlend, Crystallize, Deep, Defer, DepthGovernor, DesignArchitect, DiscoveryProtocol, DistinguishingTest, DogfoodFirst, Doubleloop, EmpathySim, EmpiricalTest, Endsettle, EpistemicROI, ErrorCostAsymmetry, EventReact, Experiment, ExploreExploit, Falsification, FixPathTriage, FractalIntelligence, Futurestretch, GeneralProblemSolvingProtocol, HorizonReview, HypothesisEngine, HypothesisLadder, Jester, Kairos, Ladderdrop, LatentWander, LateralOptimization, Leapdebt, Leapsettle, Limitlisten, ManifestPlanning, MarginalValueRule, MentalSim, MetaCheck, MetaProtocols, Metricmirror, MintWhenFriction, NoiseInjection, Novelty, OODA, OpportunityCost, OptimalStop, Optimize, PURE, PUREBrainstorming, PURECheck, PUREOptimization, Parallelize, PerspectiveEnsemble, Plateauturn, PolymorphicSolver, PracticeResponseReview, PreMortem, Prioritize, ProblemFramer, PurposeLinkReview, Reachsense, RedTeam, Reflex, RegretMinimization, RepresentationSwap, Retry, RigorousSolver, Roadmap, RootSolver, SacrificialProbe, Satisfice, Silence, Simulation, Solver, StalledDirectionReview, SteelmanFirst, StopDrill, Strategy, SunkCostIgnore, TensionHold, ThinSlice, TimeboxThink, TradeOff, UncertaintyMap, Valueprobe, Wonderdwell, WorldReversible.

Society Layer (156 patterns) Multi-agent coordination.

- **Coordination** (12): Compromise, Consensus, ConsensusFinder, Delegate, Disband, Elect, IdentityHandshake, LazyConsensus, OntologyHandshake, Rally, Resonate, Vote.

- **Dynamics** (20): CumulativeCulture, DemographicTransition, DensityRatchet, DiminishingComplexityReturns, ExitCage, Focalpeer, FreeRiderProblem, InnovationDiffusion, Linkfade, MalthusianTrap, Quietsever, Recifluence, Scaleasynchrony, SecurityDilemma, Sideinherit, StructuralDemographicCycle, Sunbound, SurplusBound, Twinflow, Warsift.

- **Economics** (9): AtomicBid, AttentionMarkets, Award, Bid, ContinuousResourceAuction, ExchangeRate, Gardener, ValuePeg, Yield.

- **Governance** (24): Absentseat, AnchorDrop, Burdenlight, Consentrenew, Constitution, Coursecandor, Dissentkeep, DocumentedOverride, Futureclaim, Goalweave, Hearshift, Localhinge, Mandatune, Powerreturn, Promisehinge, Reasonbridge, Repairmand, Responsibility, Ruleflip, SolverTree, Successiongift, Trialhaven, UniversalSolverTree, WorldTransparent.
- **Protocols** (91): AdversarialProof, AgentDiscover, AgentProtocol, AgentSandbox, AmbiguityResolution, BearerToken, Belongbend, BoundedTask, Canary, Certaintyclap, ChannelCheck, Cocapacity, Coflourish, ConfusedDeputy, ContextSwitch, Correctionwelcome, Costeer, CounterfactualAnchor, DataMinimization, DeliberativeAlign, Deploy, Discover, Disputelift, DissentSeek, Dissentflatten, Doubtshelter, DriftWatch, EbbFlowSync, EjectionSeat, EvaluatorOptimizer, ExpiringToken, FabricSharding, FeatureFlag, Framecapture, GenealogicalTrace, GlacialVault, Global, GracefulDegradation, Handoff, HeldRelease, IntentGap, InternalConsistency, InvariantFilter, LatticeCommit, Meaningpause, MemeticSeed, ModestClaim, MonotonicCounter, Mosaicmap, Nuancefilter, Nucleate, OptimisticSolver, Oracle, OrchestrationLoop, OsmoticFilter, PatternSketch, PermissionEscalate, PhasedRefinement, Premisebridge, PromiseGraph, PromptChain, PropheticQuorum, QuorumPulse, RealizationProtocol, Reasoncommons, ReceptivityGate, ReversibilityCheck, Robustness, Rollout, RolloutWatch, RootHashGossip, ShoutWhisper, SignalReflection, Silencevote, SolverNode, SomaticMarker, SpectralTune, StateLock, Stigmergy, StructuralCoaching, SynergisticMode, Taper, ThreeLevelCollision, TieredAccess, ToolDiscovery, TranslationProxy, UniqueHandle, UptakeAsGround, UptakeOverTimestamp, WorkerMode, Workflow.

C Encoding Rules

This appendix gives the exact encoding rules that Section 3.3 summarizes.

The current implementation uses canonicalization v3. Its node encoding is that of v2 (semahash 0.3.0), which domain-separates every Merkle node with an explicit type tag. Strings undergo NFC normalization followed by whitespace normalization and are hashed as SHA-256($\mathrm{s} :\|$normalized UTF-8 text); primitive JSON values as SHA-256($\mathrm{p} :\|$v2 primitive spelling); lists as SHA-256($\mathrm{l} :\|H(L_1)\|H(L_2)\|\dots$); and dictionaries as SHA-256($\mathrm{d} :\|H(k_1)\|H(v_1)\|\dots$), with entries sorted by normalized key and with normalized-key collisions rejected fail-closed. Dependency maps receive one additional canonicalization step, which keeps each alias paired with its target. A pairing is *implied* when the target's handle is well formed, the alias and the handle are equal after lower-casing ASCII letters and removing underscores and hyphens, and no other alias name anywhere in the card folds to the same string; it is then keyed by the lower-cased handle, as in v2. Every other pairing is kept under the reserved key `@bindings` of its category, as a map from alias to reference. An alias that string normalization would change is rejected. A card whose pairings are all implied therefore has the same digest under v2 and v3. These tags prevent structurally distinct JSON values—for example the string `"1"` and the number `1`, or a two-element list and a one-entry dictionary—from sharing the same hash input. The normative encoding specification defines the exact whitespace set, numeric spelling, lowercase ASCII-hex child digests, key ordering, and runtime compatibility boundaries; interoperability vectors for the node encoding and the dependency rule supply exact preimages and rejection cases.

The two vocabulary-wide roots of Section 3.3, the semantic-set root and the catalog root, are computed as follows. Both use scheme-specific leaf framing, defined in Section 6 of the normative specification, followed by the Merkle Tree Hash construction of RFC 9162 [14]: leaves are SHA-256$(0x00\|d)$, internal nodes are SHA-256$(0x01\|L\|R)$, and a non-power-of-two list is recursively split at the largest power of two below its length. Thus an unpaired subtree is never duplicated. Sorting raw digests is the rule used here for unordered-set semantics—Certificate Transparency itself receives an already ordered log—while catalog entries are ordered by their handle bytes.

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] Tobias Kuhn and Michel Dumontier. Trusty URIs: Verifiable, immutable, and permanent digital artifacts for linked data. In *The Semantic Web: Trends and Challenges (ESWC 2014)*, pages 395–410. Springer, 2014. Graph-level content hashes embedded in dereferenceable identifiers for nanopublications.

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] Ben Laurie, Eran Messeri, and Rob Stradling. RFC 9162: Certificate transparency version 2.0, 2021.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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).

[21] 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.

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

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

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

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

[26] 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'.

[27] 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.

[28] 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.

[29] 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'.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] InChI Trust. InChI FAQ. `https://inchi-trust.info/wp/download/104/inchi-faq.pdf`. Section 2.7: InChIKey. Accessed September 10, 2026.

[38] UniProt Consortium. UniParc Sequence Archive. `https://www.uniprot.org/uniparc`. Stable unique sequence identifiers. Accessed September 10, 2026.

[39] 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.

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

[41] 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.

[42] Samuele Marro. Agora Protocol for Two-Party JSON Exchanges. `https://agoraprotocol.org/docs/protocol/specification`, 2025. Working draft, January 19, 2025; sections 6 and 8.

[43] 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.

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

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

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

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

[48] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. arXiv:2512.24601, 2025.

[49] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. arXiv:2305.16291, 2023.

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

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

[52] 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.

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

[54] 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.

[55] 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.

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

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.