Fractal Intelligence

Conceptual Decomposition as Problem-Solving Infrastructure

Henrik Westerberg
henrik.westerberg@emergentwisdom.org

April 7, 2026

Abstract

Current multi-agent AI systems organize large language models the way we organize human corporations: by role, function, or task. This approach traps reasoning inside domain-specific silos, preventing an agent solving a coding problem from sharing architectural insights with an agent diagnosing a medical patient. Fractal Intelligence proposes a fundamentally different routing primitive: organizing agents by universal cognitive concepts. Instead of passing work to a "Financial Analyst" or "Software Engineer," the system decomposes problems using a strict four-test algorithm—combining classical decomposition criteria like MECE with Aristotelian universality—until it isolates pure, domain-agnostic conceptual operations. These operations are routed to specialized concept-solvers linked via a content-addressed registry (Sema). Because solvers are defined by universal concepts rather than domain-specific tasks, they are naturally reused across entirely unrelated fields. In a prototype simulation of 100 problems across 20 unrelated domains, the algorithm produces a shared conceptual structure of 456 nodes with 65% cross-domain reuse—demonstrating that a single `StabilityRegulationSolver` is routed through as a shared conceptual node by problems as different as macroeconomic currency crises and psychological emotional regulation—a topological convergence of the decomposition algorithm rather than a claim of executed cross-domain problem solving. By replacing role-based routing with concept-based routing, Fractal Intelligence introduces two architectural mechanisms: "Frame Errors," where a specialist's rigid rejection forces the system to structurally rethink its understanding of a problem, and a foundation for joint training, where specialized models can continuously compound reasoning improvements across otherwise unrelated domains.

1 Introduction

Current multi-agent frameworks naturally decompose work the way human organizations do: assess the problem, identify the necessary tasks, break them into steps, and assign specialists. This pattern has sustained companies and governments for centuries, and modern agent frameworks faithfully replicate it. But replicating it in machines inherits two distinctly human limitations that AI can escape.

The first is convention. Humans follow step-by-step processes shaped by how organizations have historically managed work. These processes thrive because they coordinate human teams effectively, and successful practices propagate across industries. One path for AI agents is simply to replicate these processes faster and at larger scale. It would work. But machines have the ability to do something fundamentally different: decompose from the structure of the problem itself rather than from the conventions that grew up around it.

The second is inference. Humans do not write down their thought processes. Expertise is largely implicit: pattern matching, norms, intuitions accumulated over years of practice. We rarely perform deliberate, step-by-step reasoning on every dimension of a problem; we approximate, using what Kahneman calls System 1 [1]. Machine learning replicates this by training on vast text corpora, learning the same implicit patterns. It works. But machines can also do what humans rarely sustain: the slow, deliberate, exhaustive reasoning that Kahneman calls System 2, applied to every dimension of a problem independently, at full depth, every time.

Machines do not tire the way humans do. They do not lose focus on step forty-seven of a hundred-step verification. They can hold the full context of a problem domain in memory while systematically checking every dimension against every constraint. What they lack is the structure to work inside. Create the conditions for elaborate, deliberate reasoning to occur on every dimension of a problem, and you increase intelligence. This paper proposes an architecture that creates those conditions. Instead of asking "how would a human solve this?", we ask: "what is this problem made of?"

Consider creating a restaurant. The conventional approach is a sequence of tasks: research the market, secure funding, find a location, hire a chef, design a menu, build out the space, obtain permits, launch marketing, open. Each step follows the last. This is how restaurants have always been built, and it works.

A conceptual decomposition instead asks two different questions. First: what does it mean to *create* something? This decomposes into intent, form, realization, fit, and viability—dimensions that apply to creating anything, whether a restaurant, a curriculum, or a building. Second: what *is* a restaurant? This decomposes into nourishment, experience, flow, economics, and identity—the orthogonal dimensions of the thing itself. Remove any one and the concept degrades: remove nourishment and it becomes a bar, remove experience and it becomes a cafeteria, remove flow and it descends into chaos. Tasks emerge at the intersection of these two decompositions, but the conceptual structure is what persists and transfers.

Once decomposed, each dimension gets its own solver, isolated behind its own boundary, operating within its own context. Flow—how people and resources move through space and time—follows the same logic in a restaurant, a hospital, a transit system, and an irrigation network. We call this Fractal Intelligence.

1. Take any problem.
2. Ask what it is actually made of—not the steps to solve it, but the irreducible dimensions of the thing itself.
3. Give each dimension its own solver with its own boundary.
4. Let each solver do the actual cognitive work rather than inferring the answer.
5. Dimensions that recur across unrelated problems become permanent infrastructure—reasoning highways.
6. The infrastructure compounds: each invocation sharpens the solver's capacity.

The underlying philosophy is not new to human civilization. A hospital decomposes into diagnostics, surgery, pharmacology, rehabilitation—each a department, each with its own specialists. A patient arrives, is routed to the relevant conceptual structure, is treated, and leaves. The patient is consumed; the hospital remains. Human civilization discovered this organizational intelligence informally, through centuries of institutional evolution. Fractal Intelligence formalizes it for the machine era: the same patterns, made explicitly composable, content-addressed, and persistent.

1.1 Fractal Intelligence

It is many agents collaborating, but through the uniform contract and conceptual boundaries, the system acts as one intelligence. Because the decomposition is conceptual, not tied to any specific domain, the architecture is general enough to accept any problem and route it through the appropriate structure. From the outside, it behaves as a single general-purpose problem solver. From the inside, it is specialists all the way down.

> **Fractal Intelligence** *is the expansion of cognitive capability through the recursive decomposition of concepts into contract-bounded sub-concepts, where the conceptual structure—what the ability is made of—persists as a reusable, composable pattern that improves through use.*

The foundation is first-principles thinking. At every node, the decomposition asks the same question: what are the irreducible dimensions of this concept? The structure that results is fractal in the precise sense: the same five-surface Solver Contract governs every level. You can zoom into any node and find the same pattern repeating at higher resolution, just as you can zoom into a Mandelbrot set and find self-similar structure at every scale. The resolution increases; the architecture does not change.

No individual node needs to be generally intelligent. The nodes are specialists. But the composition of specialists through the contract produces capabilities that no individual node can reach alone, and these capabilities compound with use. The protocol is general. The intelligence lives in the composition.

The interconnection deepens further under joint training (Section 8). In a standard agent network, agents communicate but remain independent; one agent's learning does not shape another's. In the solver tree, the acceptance gates produce a shared training signal: each node's learning depends on what the others do. The generator learns what the verifier accepts. The verifier learns what the taxonomist considers novel. The routing model learns which paths work for which problem types. Under joint training, the topology itself co-evolves with the weights: nodes split when learning reveals they are doing two jobs, and new specialists emerge as the system encounters new problem classes. The result is not a network of independent agents but a single interconnected system where the nodes reinforce each other through use.

1.2 What This Architecture Adds

How do we make the above a reality? The architecture adds six mechanisms that, in combination, provide the structural foundation:

Self-similar contracts. Every node (orchestrator, specialist, leaf) exposes the same five-surface contract (Section 4). A leaf solver and a thousand-node sub-tree look identical to the caller. Because the interface is uniform, a sub-tree that solved a problem at depth 7 is importable at depth 2. Without this uniformity, the other mechanisms lose their foundation.

Gates that force conceptual restructuring. The acceptance gate rejects inadequate work with a typed, structured explanation of what specifically failed. The upstream solver must structurally reframe its approach—not retry the same thing, not soften its claims, but restructure. This is the mechanism that produced the retirement system result described in Section 4.6: the boundary did not catch a bad answer; it forced the solver into a different ontological category.

Cognitive isolation enabling productive extremes. Cognitive mode is declared in the Manifest and enforced by typed boundaries. A maximally creative solver can be maximally creative because a downstream maximally strict verifier will catch what it misses. Neither compromises [2].

Economic depth governance. The marginal-value rule (Section 3.2) decides at every node whether to decompose further or stop. Depth concentrates where uncertainty is highest and stays shallow where the problem is easy.

Persistent decomposition and learning through reuse. The decomposition itself crystallizes as a content-addressed pattern importable by hash (Section 4.5). What persists is not an answer but the anatomy of a competence. Because the contract is uniform, the same solver gets called by structurally similar problems across different domains, accumulating cross-domain signal and forming reasoning highways (Section 5.2).

Self-organizing topology. The tree restructures itself based on what it learns. When a solver is doing two conceptual jobs, it splits. When two solvers are doing the same job, they merge. The prototype produced 43 restructuring events across 100 problems, driven by the data, not by a designer.

Table 1: Standard agentic pipeline vs. Solver Tree.

	Agent Pipeline	**Solver Tree**
Organizing principle	Task (a piece of work)	Concept (tasks route through it)
Identity basis	Person/Role (the "who")	Concept/Interface (the "what")
Topology	Fixed before execution	Discovered at runtime
Boundaries	Natural language, pass-through	Typed, gated (reject $\rightarrow$ restructure)
Depth allocation	Uniform across stages	Selective (marginal-value rule)
Failure signal	Propagates silently downstream	Caught at the seam; triggers reframe
Cognitive mode	Generalist at every node	Declared, enforced, separated
Output	An answer	An answer + reusable structure
Persistence	Dissolves after execution	Crystallizes as content-addressed pattern

1.3 Contributions

Our contribution is the specific claim that **decomposing problems at the conceptual level—and routing tasks through that persistent structure—changes what the system can do**. The boundary between concepts produces intelligence that cannot exist without it, through three mechanisms: productive extremes (each concept operates without compromise because the gate catches what it misses), structural reframing (rejection at a gate compels conceptual restructuring, not iteration), and persistent structure (the conceptual decomposition crystallizes and compounds through reuse).

Each mechanism has identifiable ancestors. Recursive decomposition is ancient: divide-and-conquer in algorithm design, means-ends analysis in cognitive science [3], Babbage's division of mental labor [4], Minsky's society of interacting agents [5], Parnas's modular decomposition [6]. Self-similar agent organization has two foundational ancestors: the actor model [7], which treats every entity as an actor with a uniform interface and makes no ontological distinction between coordinators and workers; and fractal manufacturing systems [8, 9], where autonomous units with uniform internal structure coordinate across factory floors. Contract-based multi-agent coordination traces to the Contract Net Protocol [10] and is actively reinvented in today's typed agent interfaces [11, 12, 13]. Persistent reasoning reuse is a concurrent theme: Graph-Memoized Reasoning [14] caches workflow traces for retrieval across tasks.

What distinguishes this work from each ancestor is the same thing: the universality test. Zwicky's morphological analysis [15] decomposes design parameters—fuel type, nozzle shape—but the parameters are domain-specific; they do not recur across unrelated fields. Fractal manufacturing systems provide self-similar organizational units, but decompose manufacturing tasks directly; this paper decomposes the conceptual structure first, and tasks route through that structure, which is why the same solvers recur across unrelated domains. Contract-based coordination validates at boundaries but does not force the upstream agent to restructure its ontology when the gate rejects—and contemporary agent protocols that organize coordination economically rather

than conceptually inherit the same gap. Workflow memoization caches execution traces (the steps an agent took), not conceptual decompositions discovered by independent analysis.

The four tests, and the universality test in particular, are the mechanism that produces domain-general sub-concepts: they select for axes that every instance of the parent concept must contain, which is why independent decompositions of unrelated domains converge on the same sub-concepts. The convergence is not designed; it is a consequence of the algorithm. This is what makes "reasoning highways" an architectural prediction rather than an engineering aspiration.

Fractal Intelligence achieves compositional intelligence through two mechanisms: (1) **The Theory of Depth** (Section 3): a theory of when and where to decompose conceptual work, culminating in the Universal Solver Tree; and (2) **The Solver Contract** (Section 4): a specification of how concept-defined Solvers compose safely through five typed surfaces and acceptance gates. These produce two essential properties: *productive extremes and emergent structure* (Section 4.6), where cognitive isolation enables each Solver to operate without compromise, and *localized learning* (Section 5), where each Solver improves at its conceptual specialty without degrading others. Together, these constitute the protocol layer for a decentralized cognitive commons (Section 7).

2 The Conceptual Decomposition Algorithm

The preceding section claims that Fractal Intelligence decomposes at the conceptual level, building the persistent structure through which tasks route. But how? What is the algorithm that takes a concept and finds its constituent sub-concepts? This section presents a first formalization of a process that humans perform intuitively (first-principles decomposition) and makes it explicit enough to be executable by large language models. The four tests and the pseudocode below are a starting point, not a final specification. The real algorithm may look very different; it may involve human judgment, iterative refinement, or hybrid human-machine collaboration rather than a single LLM pass. What matters is the structural commitment: decompose concepts first, let tasks emerge from that structure, and apply necessity, independence, universality, and completeness as the governing criteria. This section demonstrates the approach across three unrelated domains and shows that the sub-concepts that emerge are inherently domain-general, which is why reasoning highways form. This is the difference between philosophical first principles (a thinking style confined to one mind) and architectural first principles (an executable decomposition across cognitive boundaries). The former depends on the thinker's discipline to resist recombining what they have separated; the latter enforces the separation structurally.

2.1 The Four Tests

When decomposing a concept, the algorithm does not ask "what are the steps to accomplish this?" (that produces tasks). It asks: *what are the independent dimensions that must all be addressed for this concept to function?* The algorithm finds the orthogonal basis vectors of a concept space—the independent axes that span what the concept is. Four tests govern whether a candidate dimension qualifies:

Necessity. If you removed this sub-concept, would the parent concept still function? If removal causes the parent to collapse, the sub-concept is load-bearing. An economy without selection (what gets produced) is not an economy. Education without assessment (how do you know learning occurred) is indistinguishable from entertainment.

Independence. Can you change one sub-concept without necessarily changing the others? You can redesign an economy's selection mechanism (from market to planned production) without changing your theory of fairness. You can redesign an educational assessment system without

altering the content curriculum. If two sub-concepts always co-vary, they are actually one concept wearing two names—merge them or find the real axis.

Universality. Does every instance of the parent concept contain this sub-concept? Every economy that has ever existed—hunter-gatherer, feudal, capitalist, communist, post-industrial—has selection, distribution, valuation, coordination, and stability. Every educational system—Montessori, PhD programs, apprenticeships in 1400s Florence—has content, transmission, assessment, motivation, and adaptation. If a sub-concept only applies to some instances ("stock market regulation," "multiple-choice testing"), it is not a fundamental axis—it is a task inside one implementation.

Completeness. If you addressed all the sub-concepts, would you have a functioning instance of the parent concept? If something is missing, there is an axis you have not found.

The four tests have classical lineage. Necessity, independence, and completeness echo the Mutually Exclusive, Collectively Exhaustive principle formalized in Minto's Pyramid Principle [16] and earlier in standard decomposition methodology. Universality is the classical Aristotelian criterion of necessary properties for category membership, formalized mathematically in Formal Concept Analysis [17], where a formal concept's intent is precisely the set of attributes shared by all objects in its extent. The contribution here is not the criteria themselves but their packaging as a generative LLM-executable suite for recursive sub-concept screening: rather than describing the intent of an existing concept lattice, the algorithm uses universality (with the other three tests) to discover, at machine speed, the conceptual structure through which novel problems should route. A cognitive-science reader will object that natural human concepts rarely obey strict universality: Rosch's prototype theory [18] and Wittgenstein's family-resemblance critique demonstrate that everyday categories cohere around graded similarity rather than necessary-and-sufficient features. The objection is correct as a description of human concepts and is not the claim being made. Strict universality is invoked here not as a model of cognition but as an adversarial crucible: an intentionally unforgiving filter that burns away the prototype-shaped slack an LLM would otherwise smuggle into its decompositions. The point is to generate domain-general invariants on which solver reuse is structurally enforceable, not to recover the texture of human concept formation.

Then recurse: each sub-concept is itself a concept. Apply the same four tests. The recursion terminates when a sub-concept is concrete enough to be addressed directly by a single Solver—the point at which the marginal-value rule (Section 3.2) indicates that further decomposition would cost more than it yields.

Three implementation notes are critical. First, the four tests are evaluated by an LLM, which means they are probabilistic judgments, not mathematical proofs. This is appropriate: a human applying first-principles thinking also makes judgment calls about whether a dimension is truly independent or merely appears so. The algorithm formalizes the structure of the judgment, not the certainty of the outcome. Gate rejections and Pathway Memory provide the feedback loop that corrects errors over time.

Second, pure independence is the ideal; real concepts leak into each other. Distribution and Selection in an economy *do* interact. The algorithm seeks the best available cut, not a perfect one. When leakage causes downstream failures, the Frame Error mechanism (Section 5) propagates upward, signaling that the decomposition itself must be reframed. The leaky seam discussion in Section 3.1 addresses this directly.

Third, the routing sanity check prevents hallucinated decompositions from persisting. Before a decomposition crystallizes as a Sema pattern (a content-addressed specification, detailed in Section 4.5), three concrete domain-specific tasks are generated and routed through the proposed sub-concepts. If a task cannot find a clean home, there is a missing dimension. If a task belongs equally to two dimensions, they are not independent enough. This empirical grounding step

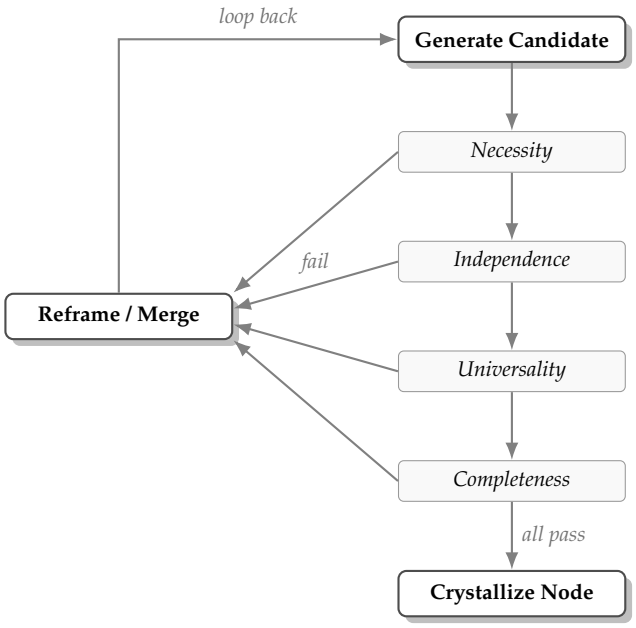

Figure 1: The four-test generative algorithm flow—a visual summary of the `DECOMPOSE_CONCEPT` algorithm given in precise pseudocode form as Figure 2. White nodes are processing steps (generating candidates, reframing on failure, crystallizing accepted sub-concepts). Gray nodes are the four screening tests. Failures at any test route through the Reframe / Merge step, which loops back to candidate generation with updated structural constraints. The two figures present the same algorithm at different levels of detail: this one shows the control flow at a glance, while Figure 2 captures the completeness retry limit, the routing sanity check, and the recursive descent.

is what prevents the algorithm from producing decompositions that are logically elegant but functionally useless.

In a Fractal Intelligence deployment, this algorithm is not executed by a human architect. It is executed by a Solver—typically the `RootSolver` or a parent Solver deciding how to decompose its own concept. Because large language models can generate candidate dimensions, evaluate the four tests, reframe when tests fail, and restructure the hierarchy at machine speed, the conceptual decomposition is discovered at machine speed and refined through use. The algorithm is the same whether a human applies it on paper or a Solver applies it at runtime; the difference is that a Solver can trial thousands of candidate decompositions and select the one whose coupling signatures are cleanest.

Each sub-concept is itself a decomposition point. "Need resolution" decomposes further into ambiguity resolution, stakeholder modeling, and intent inference—the same sub-structure whether applied to a marketplace, an essay, or a curriculum. The tasks that emerge within each are entirely different, but the conceptual anatomy is stable.

```
DECOMPOSE_CONCEPT(concept, depth_budget):
  candidates = generate_candidate_dimensions(concept)
  axes = []
  for each candidate:
    if NOT necessary(concept, candidate):  skip
    if NOT independent(candidate, axes):   merge or reframe
    if NOT universal(concept, candidate):  skip — it's a task
    axes.append(candidate)
  // Completeness: search governed by diminishing returns
  retries = 0
  while NOT complete(concept, axes) AND retries < 3:
    new_candidate = search_for_missing_dimension(concept, axes)
    if new_candidate passes all three tests:
      axes.append(new_candidate)
    retries += 1
  // Routing sanity check: empirical grounding
  test_tasks = generate_sample_tasks(concept, n=3)
  for each task in test_tasks:
    if task cannot route cleanly to exactly one axis:
      reframe axes until routing is clean OR accept with warning
  // Recurse where depth budget permits
  for each axis in axes:
    if marginal_value(axis) > theta:
      axis.children = DECOMPOSE_CONCEPT(axis, depth_budget - 1)
  return axes
```

Figure 2: The four tests (necessity, independence, universality, completeness) govern which candidates qualify as sub-concepts. The completeness search halts after diminishing returns. A routing sanity check empirically validates the decomposition before it crystallizes. Recursion depth is governed by the marginal-value rule.

2.2 Example: Decomposing the Economy

"Solve the economy" is the kind of task that seems unapproachably vague. Task decomposition produces project-management artifacts: gather data, consult experts, draft policy, implement, evaluate. Conceptual decomposition asks: what are the orthogonal dimensions that every economy must address?

- **Selection** (what gets produced?): resource allocation, innovation incentive, scarcity resolution.
- **Distribution** (who receives the output?): fairness (procedural, distributive, intergenerational), access, transfer efficiency.
- **Valuation** (how is worth determined?): price formation, externality accounting, temporal discounting.
- **Coordination** (how do actors synchronize?): information flow, trust, incentive alignment.
- **Stability** (how does the system handle shocks?): shock absorption, feedback regulation, resilience.

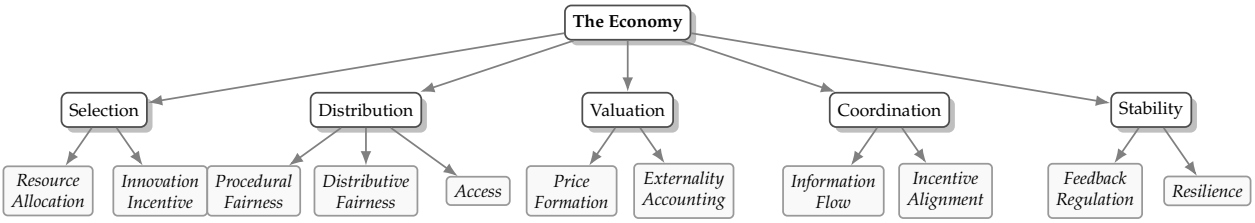

Figure 3: Conceptual decomposition of the economy. White nodes are concepts (persistent, domain-general). Gray nodes are sub-concepts that recurse further.

Apply the four tests. Remove Selection—nothing gets produced; the economy collapses (*necessary*). Remove Distribution—output has no destination (*necessary*). Can you change Selection without changing Distribution? Yes—you can move from market to planned production without altering your fairness theory (*independent*). Does every economy have all five? Yes (*universal*). Address all five and you have a functioning economy (*complete*).

That these dimensions are routinely fused in practice—selection and distribution bundled into a single governing philosophy, valuation and coordination assumed to require the same mechanism—does not make them dependent. It makes the convention visible. Assumptions that were necessary when human organizations bore the coordination overhead may not hold when the agents are machines operating behind typed contracts.

Notice what is not in this tree: raise interest rates, adjust taxes, create jobs, build infrastructure. Those are tasks. They emerge inside the conceptual structure. "Raise interest rates" is a task that lives inside Stability → Feedback Regulation. "Adjust taxes" lives inside Distribution → Fairness → Distributive Fairness. The task depends on the specific economy. The concepts are the same for every economy that has ever existed or could exist.

Now recurse one level. Distribution → Fairness decomposes into procedural fairness (is the process just?), distributive fairness (is the outcome just?), and intergenerational fairness (is it just across time?). Each is necessary, independent, and universal. A `ProceduralFairnessSolver` defined by the concept of process justice would be called not only by economic system design but by legal systems, hiring processes, hospital resource allocation, and school admissions. The concept transfers because it is a concept, not a task.

2.3 Example: Decomposing Education

- **Content** (what is worth knowing?): knowledge selection, sequencing, relevance.
- **Transmission** (how does knowledge transfer?): representation, interaction, scaffolding.
- **Assessment** (how do you know learning occurred?): measurement, validity, feedback timing.
- **Motivation** (why does the learner engage?): intrinsic drive, extrinsic structure, meaning.
- **Adaptation** (how does instruction adjust?): diagnosis, differentiation, pacing.

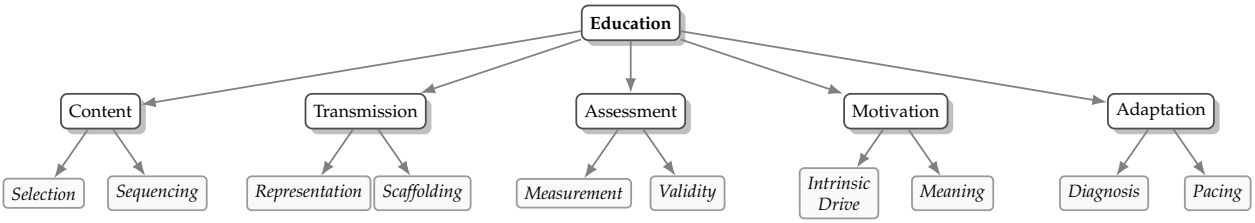

Figure 4: Conceptual decomposition of education into five orthogonal dimensions, with domain-general sub-concepts at depth two.

The four tests hold. Remove Content—nothing to learn, education collapses. Remove Assessment—no way to know if learning happened. Remove Motivation—the learner does not engage, nothing occurs. Each is independent: you can redesign assessment without altering content. Each is universal across every form of education. Together they are complete.

The sub-concepts are again domain-general. Assessment $\rightarrow$ Validity (does the test measure what it claims?) is the same concept whether applied to education, medical diagnostics, or software testing. Adaptation $\rightarrow$ Diagnosis (where is the learner?) is structurally isomorphic to medical triage (where is the patient?) and debugging (where is the fault?).

2.4 Example: Decomposing Game Design

- **Agency** (what can the player do?): action space, consequence, expression.
- **Challenge** (what resists the player?): difficulty calibration, uncertainty, constraint.
- **Feedback** (how does the player know their state?): immediacy, clarity, richness.
- **Progression** (how does the experience change?): mastery curve, revelation, pacing.
- **Coherence** (why does it hold together?): internal consistency, thematic unity, aesthetic integrity.

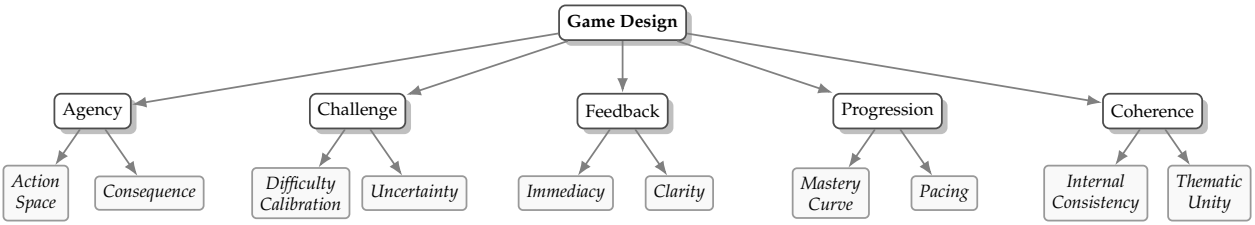

Figure 5: Conceptual decomposition of game design into five orthogonal dimensions. The necessity test confirms each is load-bearing.

Remove Agency—the player is a spectator, not a game. Remove Challenge—no resistance, it is a toy. Remove Feedback—the player cannot learn, it is noise. Universal: chess, a modern video game, poker, tag in a schoolyard—all have these five dimensions.

2.5 Cross-Domain Convergence: Why Reasoning Highways Form

Across these three unrelated domains, the same sub-concepts recur:

Feedback appears as Assessment in education, Feedback in game design, and Feedback Regulation inside Stability in economics. A Solver whose concept is "How does an agent learn its current state through signals?" would be called by all three domains.

Incentive alignment in economics and *motivation* in education are structurally isomorphic: both decompose into intrinsic drivers, extrinsic structures, and meaning-making. A Solver that reasons about why actors engage would serve both domains.

Adaptation in education (diagnosis → differentiation → pacing) and *coordination* in economics (information flow → response → synchronization) share the same structure: sense the state, adjust the path, manage the tempo.

Validity in educational assessment, *internal consistency* in game design, and *procedural fairness* in economic distribution all address the same underlying concept: does this system actually do what it claims to do?

The four tests explain why. Because they select for necessity, independence, universality, and completeness, the sub-concepts that survive are the most abstract, most domain-general dimensions of the parent concept. They recur across domains because they are the irreducible components of how things work. The uniform contract routes structurally similar problems to the same Solvers regardless of which domain generated them.

Convergence enables compounding. Because the same Solver handles structurally similar sub-problems from different domains, it accumulates cross-domain signal. An `IntentSolver` that has resolved ambiguous intent for fifty different creations should get better at interpreting intent for the fifty-first. This is learning through repetition: the conceptual capacity sharpens because it is exercised across many instances. The bolder claim is that the signal transfers across genuinely different domains: an `UncertaintyPropagationSolver` that has modeled chaos in weather systems has learned something about uncertainty propagation *as a concept* that may help it model chaos in financial markets. Whether this cross-domain transfer actually operates at scale remains the central open question (Section 11).

The algorithm also explains where tasks emerge. Tasks live at the leaves of the conceptual tree—the point where a sub-concept is concrete enough to be addressed directly. "Design a multiple-choice exam" is a task inside Assessment → Measurement for a specific course. The tasks are domain-specific; the concepts above them are not.

2.6 When Decomposition Goes Wrong

Two archetypal failures deserve attention. A *leaky seam* looks clean on paper but implicit assumptions bleed across the boundary. If the acceptance criteria for a `SelectionSolver` implicitly encode a distribution assumption—maximize GDP, which presupposes market-based distribution—then the distribution philosophy has bled into the selection boundary. The remedy is to strengthen the interface so the assumption must be explicitly typed. A *misaligned cut* separates variables that must co-vary; the remedy is to reframe until the dominant uncertainty sits on one side.

The cost of a bad cut is severe. If a parent solver makes a misaligned cut at the root of the tree, all downstream solvers inherit a flawed ontology, wasting compute optimizing the wrong variables. This is why the Frame Error feedback mechanism (Section 5) is critical: it allows leaf-node friction to invalidate root-node decompositions before the flaw poisons the entire tree.

2.7 The Stability of Conceptual Structure

Conceptual infrastructure is built once and refined through use. The economy has had Selection, Distribution, Valuation, Coordination, and Stability for ten thousand years. Education has had Content, Transmission, Assessment, Motivation, and Adaptation since the first teacher sat with the first student. These dimensions do not change when a new task arrives. They barely change across centuries.

The algorithm is run to discover the infrastructure. Then tasks flow through it—thousands of tasks, millions of tasks, all routing through the same conceptual tree. The algorithm is re-run occasionally: when Pathway Memory reveals that a sub-concept is doing two jobs and should split, or when a genuinely new dimension is discovered that the original decomposition missed. But this is rare. The structure is stable because concepts are stable. The learning happens within the infrastructure, not by redesigning it. Routing is partly reasoned from first principles (the four tests determine the conceptual structure) and partly learned from experience (which paths work for which problem types). Each node sharpens at its specific domain through whatever mechanism is available behind its contract. When the data reveals a node is doing two jobs, the topology splits. Section 5 develops the full learning mechanism.

This is what makes it infrastructure in the literal sense. You build roads once. You maintain them. Traffic flows. You do not redesign the road network for every car. The conceptual decomposition algorithm builds the road network. Tasks are the traffic. The network densifies at bottlenecks and occasionally adds a new route, but the basic topology persists.

This also answers the orchestration overhead objection. Running the decomposition algorithm is not cheap—it requires LLM calls to generate candidates, evaluate the four tests, and recurse. But you run it once per domain, not once per task. The amortized cost across thousands of tasks is negligible. The expensive part is discovering the conceptual structure. Using it is cheap.

2.8 Emergence in Reverse

The architecture produces something that resembles emergence running in reverse. In conventional emergence, simple local rules produce complex global behavior that nobody designed—flocking from boids, markets from traders, consciousness perhaps from neurons. The surprise is at the top: you define the micro and the macro astonishes you.

This architecture inverts the direction. You define abstract concepts at the top—"stability regulation," "incentive alignment," "feedback regulation"—and recurse downward through the decomposition algorithm. What surprises you is what appears at the leaves and between the branches. Nobody designs the connections in advance. An `IncentiveAlignmentSolver` created for economics may turn out to serve dozens of unrelated domains. A `StabilityRegulationSolver` built for one context may route to a completely different one. Those connections fall out of top-down conceptual recursion, the same way flocking falls out of local rules—except the surprise is at the bottom, not the top. You defined the macro and the micro astonished you.

The reasoning highways are the emergent phenomenon. They were not designed into the system. They appeared because the four tests, applied independently to unrelated domains, converged on the same irreducible conceptual dimensions. The analogy to biological emergence is imperfect—this is a designed system, not a natural one—but the key property is shared: the interesting structure was not specified by anyone. It fell out of the process.

3 A Theory of Depth

Section 2 defines how to find the right decomposition. This section governs how deep to go. At every node, a Solver faces the same binary choice: solve directly, or decompose further? This single decision, repeated at every level of abstraction [6, 4], produces a *solver tree*: a hierarchy where each node applies the same contract regardless of substrate. There is no separate "worker" type and "orchestrator" type: only Solvers.

The *Universal Solver Tree* is their aggregate: the total topology of all solver trees across all agents in the system. Its foundational claim is that intelligence has two engines: vertical optimization through decomposition, and lateral discovery through reframing. Most systems stop far too early. There is always a more granular way to decompose a concept: more variables to isolate, more sub-structures to specialize, more dimensions to separate. The architecture makes this depth practical: any node can be "zoomed into" by spawning sub-solvers, with the marginal-value rule governing when the additional depth pays for itself.

3.1 Selective Depth

Under realistic cost models, returns to depth are diminishing while costs accumulate. Moreover, uncertainty is unevenly distributed. Uniform depth wastes budget on easy spans and starves the hard ones. The remedy is selective depth: concentrate refinement on sub-concepts with high uncertainty, brittle dependencies, or large impact. A *hard seam* is a boundary whose failure collapses the whole plan. Depth should preferentially act at hard seams—the payoff is exponential, eliminating not one branch but the entire sub-forest below it.

Unlike monolithic models that scale test-time compute opaquely through hidden chain-of-thought traces (e.g., OpenAI's o-series), a solver tree enables explicit, asymmetric compute scaling: dynamically routing inference budgets to hard seams, resolving shallow spans instantly, and producing an auditable graph of work rather than a monolithic token stream. Where existing approaches scale reasoning depth within a single context window, Fractal Intelligence proposes an orthogonal scaling axis: making reasoning *composable* across typed boundaries and trust domains, where cognitive mode is isolated and failure forces structural reframing.

3.2 The Marginal-Value Rule

The economic governor for depth is a ratio test whose lineage runs through information value theory [19], ecological foraging [20], and rational metareasoning [21, 22]: deepen if and only if the expected improvement divided by the incremental cost exceeds a domain-appropriate threshold θ. Formally, at any node n:

$$\frac{\mathbb{E}[V(n_{\text{deep}})] - V(n_{\text{shallow}}) + U(n)}{C(n_{\text{deep}}) - C(n_{\text{shallow}})} > \theta\,. \tag{1}$$

De Sabbata et al. [23] applied this directly to LLM reasoning depth. This paper extends the principle from a single model's reasoning depth to a tree of contract-bounded solvers, governing not just how long to think but where to grow the hierarchy. The bonus term $U(n)$ is an exploration credit inversely proportional to the density of Pathway Memory at node n, in the spirit of the upper-confidence-bound rule of Auer, Cesa-Bianchi, and Fischer [24]: nodes the system has rarely visited receive a temporary subsidy that converts the rule from pure exploitation into an exploration–exploitation trade-off, with the cold-start probe phase of Section 11.6 as its practical implementation.

In practice, an orchestrator implements this through a control loop:

```
ALLOCATE-DEPTH(branches, budget, theta):
  while budget > 0:
    ratios <- empty map
    costs  <- empty map
    for each branch b in branches:
      (dV, dC) <- b.Consult("estimate_next_step")
      ratios[b] <- dV / dC
      costs[b]  <- dC
    b_star <- argmax(ratios)
    if ratios[b_star] < theta:
      break
    b_star.Execute()
    budget <- budget - costs[b_star]
  return collect_results(branches)
```

Figure 6: The marginal-value rule as a control loop. The Consult surface makes the economic governor computable.

How does the Consult surface produce reliable estimates? The orchestrator does not rely on zero-shot self-evaluation. Estimates are grounded in Pathway Memory (Section 5.1): each Solver maintains structured logs of past executions, recording which problem classes it has seen, what depths yielded improvement, and at what cost. A query like `estimate_next_step` returns a lookup: "For problems structurally similar to this one, deepening with pattern Y has historically yielded a 12% quality improvement at cost $0.05." Where historical data is unavailable, the estimate defaults to a calibrated prior; when neither is available, the system allocates a fixed exploration budget to shallow probes before committing depth.

A flat marginal-value trajectory is not just a halt signal; it triggers a reframe. When continuing deeper yields no gains, the system shifts from vertical decomposition to a lateral jump.

3.3 Topology Freedom

The contract constrains what crosses boundaries (typed artifacts and acceptance gates) but not how Solvers are arranged or sequenced. This freedom is not merely an engineering choice; it is a structural consequence of the payload. A task-oriented pipeline typically passes an *instruction* ("do this specific thing"), which tends toward a fixed, sequential topology. A conceptual Solver passes an expression of *intent* ("resolve this concept"). Intent admits multiple fulfillment strategies. Because a parent Solver passes intent, it cannot prescribe the child's execution path. The topology must therefore emerge from the evidence. The same contract supports radically different execution topologies:

Sequential chains: output from Solver A to Solver B to Solver C. *Evidence-gated routing*: a parent examines a child's result and decides where to send it based on what the result contains—not based on its position in a sequence. *Fan-out and fan-in*: parallel solvers pursuing competing hypotheses, synthesized at a typed boundary. *Cycles*: a creative Solver and a verification Solver cycle repeatedly until the acceptance gate clears or the marginal-value rule halts. *Flat orchestration*: a parent routes among peers based on evidence, no tree depth. *Recursive nesting*: any of the above inside a single Solver that presents a single typed interface.

While we retain the name "Universal Solver Tree" for its conceptual clarity—decomposition is top-down and tree-like in origin—the actual data structure is a directed graph: fan-in, deduplication,

and cycles mean $|E| > |V| - 1$ in practice (the prototype simulation produced 456 nodes with 582 edges).

The key principle is that the parent Solver's routing decision is the intelligence at that level. The routing mechanism is typically implemented as a combination of Manifest matching (comparing the intent against declared concepts via vector similarity over Manifests) and Pathway Memory lookup (which solver has historically performed best on this class of intent). The conceptual structure is the blueprint; the routing through it emerges from each problem.

3.4 Jumps and Reframing

When vertical decomposition bottlenecks—the marginal-value rule detects a flat improvement trajectory—a solver cannot simply "think harder" within the same invariants. It must perform a lateral jump: reframing the problem so that the space of solutions changes.

The primary mechanism is the Discovery Protocol (Section 6.4) applied to interpretations rather than solutions. A population of reframing solvers attacks the stuck problem from maximally different angles. A `WhyChainSolver` asks "why?" repeatedly until the stated problem dissolves into an underlying need. A `ConstraintInversionSolver` systematically negates each assumed constraint to discover which ones are load-bearing and which are inherited assumptions. An `OrthogonalInsightSolver` constructs an alternative physics—deleting a concept the original framing took for granted—and solves the problem from alien axioms, then extracts the mechanism back to reality [25]. The retirement system example in Section 4.6 demonstrates this: alien physics deleted the concept of accumulation, forcing the solver to build from different axioms entirely.

These candidate reframings flow into a `TaxonomistSolver` that maintains an exhaustive graph of interpretations, repeatedly demanding structurally novel framings—not variations on what exists, but framings that open regions of the solution space the current decomposition cannot reach. When the taxonomy saturates (declining rate of new nodes), the `ReduceSolver` selects the framing that maximizes downstream decomposability: which reframing produces the richest, most independent sub-problems? The user may also be brought back into the loop—the typed output of the reframing phase is a structured menu of interpretations, each with its trade-offs made explicit, that the user can select from or refine.

The meta-perspective is maintained through parallel solvers at different levels of abstraction. Parent nodes see how the parts relate; children see how the parts work. Critically, this is bidirectional: a child solver may discover something that reshapes the parent's understanding—the Feedback surface is the channel through which children inform parents. A Frame Error at a child propagates upward not as a failure but as evidence that the parent's decomposition needs revision.

4 The Solver Contract

The Solver Contract specifies how concept-defined abilities compose across boundaries. The distinction between Solver and agent is architectural: an agent is an implementation (a model, a program, a human), while a Solver is the interface that implementation presents to the system. The same five-surface contract may be implemented by a lightweight script, a frontier LLM, or a swarm of thousands; the caller interacts with each identically.

A Solver specializes through three complementary mechanisms: *weight adaptation* (fine-tuning for a narrow problem class), *tool composition* (invoking external capabilities), and *execution harness* (a deterministic protocol guiding a general model through a domain-specific process). The contract is indifferent to which mechanism operates behind the interface. Section 4.4 develops this opacity in detail: a single agent may implement many Solver contracts, a single Solver may orchestrate many

agents internally, and the lifecycle from general-purpose hat to specialized system is governed by the same economic rule that governs depth.

The contract's intellectual ancestry is precise. Hoare [26] established verification through preconditions and postconditions. Meyer [27] operationalized Design by Contract. Liskov and Wing [28] extended behavioral subtyping. Findler and Felleisen [29] showed that contracts compose for higher-order functions—the recursive pattern Solvers require. The Solver Contract adds something absent from this tradition: the boundary is not merely a validator but a cognitive act that forces upstream restructuring when the gate rejects—the ontological accommodation described in Section 3.4. Every Solver exposes the same minimal five-surface interface (Table 2).

Table 2: The five universal surfaces of the Solver Contract.

Surface	Core Question	Purpose
Manifest	What can you do?	Declares the concept this Solver reasons about, its operations, input/output types, and any extended surfaces. Enables discovery and safe composition.
Execute	Can you do the work?	Takes a typed `Task`, returns a typed `Result`. The surface that performs conceptual work.
Consult	How would you approach this?	Open consultation: cost/quality estimation, clarification, rationale requests, context-sharing. Enables the marginal-value rule.
Verify	Why should I trust you?	Post-execution assurance: test vectors, reproduction recipes, verifiable evidence.
Feedback	How did you do?	Receives structured evaluation from downstream, or signals a Frame Error requiring a lateral jump. Enables localized learning.

The first two surfaces are mandatory; the remaining three are optional but strongly recommended at hard seams.

4.1 Gates That Redirect

At each boundary between Solvers, three things happen: (1) mode is declared (the Solver explicitly states its cognitive stance); (2) output is typed (the artifact crossing the boundary has explicit structure); (3) acceptance is gated (an `AcceptSpec` defines what must be true for the artifact to cross). The strictness of the gate scales with the seam. At soft seams (low-stakes internal handoffs), acceptance may be a lightweight shape check. At hard seams—structurally critical junctures or boundaries crossing trust domains—acceptance is non-compensatory: if any critical criterion fails, the artifact is rejected regardless of other scores.

These create *seams that bite* [30]: boundaries tight enough to maintain distinct cognitive modes, yet flexible enough to preserve internal freedom. By enforcing isolation, the architecture prevents the mode collapse common in monolithic models.

This does not mean cognitive mode blending is always harmful. During divergent phases (e.g., the variance phase of the Discovery Protocol), soft seams intentionally preserve permeable boundaries. Hard seams activate during convergent phases, when candidates must survive adversarial scrutiny. The AcceptSpec's strictness is a function of the cognitive phase.

Deterministic gates verify properties mechanically: schema conformance, hash matches, unit tests. These are immune to model failure. *Probabilistic gates* rely on LLM evaluation. These inherit evaluator vulnerabilities: sycophancy, laziness, and *evaluator collusion*—where generators and gate evaluators from the same model family share blind spots. The mitigation is structural: adversarial

architectures at hard seams, with evaluators from different model families, ensemble juries, or dedicated red-team solvers. The Discovery Protocol (Section 6.4) demonstrates this concretely: the `TaxonomistSolver` must be drawn from a different cognitive mode than the generators it evaluates.

A third outcome complements pass and reject: *override-with-documentation*. When an artifact fails quantitative criteria but the failure reflects the limits of the criteria rather than the limits of the work, the gate permits passage with a typed justification citing what the criteria missed, why it matters, and how to test the override. This prevents deadlocking on problems where quantifiable criteria cannot capture everything. The Ethical Reasoning Protocol (Section 6.6) instantiates this mechanism.

The Solver Contract specifies a *floor*, not a ceiling. The acceptance gate verifies the typed artifact (the minimum that must be true) but does not limit what else flows across. A solver can pass its full reasoning tree, rejected alternatives, and confidence distributions alongside the verified output—by reference rather than as raw text, avoiding context-window bloat—and the downstream solver can use whatever helps.

4.2 The Transaction Model

The fundamental interaction is a transaction. A Solver receives a `Task` object and returns a `Result` object:

Table 3: The Task–Result transaction.

Task Fields	`operation`, `inputs` (typed), `acceptance_criteria` (`AcceptSpec`), `budget` (time/cost/compute limits)
Result Fields	`outputs` (typed artifact), `status` (`success/partial/fail`), `stop_reason` (`completed/budget/quality`)

This forces handoffs to be structurally explicit. A Task payload crossing a hard seam:

```
{
  "operation": "sema:Optimize#0921",
  "inputs": {
    "mechanism": "Solid-State Battery Electrolyte",
    "target_metric": "ionic_conductivity"
  },
  "acceptance_criteria": {
    "schema": "sema:AcceptSpec#e4f1",
    "criteria": [
      {"metric": "conductivity_mS_cm", "comparator": ">=", "threshold": 1.0},
      {"metric": "thermodynamics", "status": "pass"}
    ]
  },
  "budget": { "max_compute_usd": 0.50 }
}
```

Figure 7: Example Task payload crossing a hard seam, with typed inputs, acceptance criteria referencing a Sema hash, and budget.

The `schema` field references a Sema content-addressed hash (`sema:AcceptSpec#e4f1`), guaranteeing that the acceptance criteria are cryptographically verifiable. The `status: partial` option enables the anytime property: a Solver that exhausts its budget returns the best result so far.

Figure 8 shows the full lifecycle of a transaction across a hard seam.

4.3 Recursive Composition

Any Solver may decompose its Task into sub-Tasks and delegate to child Solvers, applying the marginal-value rule at each level (Figure 9). Constraints are inherited fully and monotonically: if the root Task specifies "no network access," every descendant inherits that constraint.

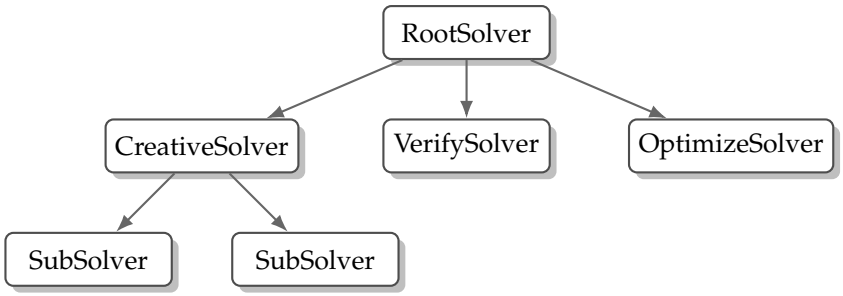

Figure 9: Recursive decomposition. Each Solver may delegate to children, applying the same contract at every level. Depth is governed by the marginal-value rule.

The apex case is the `RootSolver`: a Solver whose Task is an open-ended goal and whose cognitive operation is triage—determining what kind of problem this is and routing it. The `RootSolver`'s Pathway Memory is the architecture's most consequential site of compounding: it is the only node that sees how all problems enter the system.

The contract is invariant across five levels of ceremony: (1) natural language coordination, (2) pattern-enriched dialogue using Sema patterns, (3) lightweight Solvers with partial surfaces, (4) rigorous Solvers with full contracts and non-compensatory gates, (5) recursive orchestration with deep trees. Companion experiments suggest the largest gains may be front-loaded: the biggest improvement occurs between unstructured and pattern-enriched coordination [31]. The five surfaces guarantee composability in three coordination modes: *hierarchical* (parent calls child), *peer-to-peer* (Solvers with compatible extended endpoints engage directly), and *stigmergic* (Solvers interact through a shared medium [32] rather than direct calls).

4.4 What Sits Behind a Solver

The five-surface contract specifies what crosses the boundary, not what sits behind it. This architectural opacity is deliberate and admits three deployment configurations:

- *One agent, many Solvers.* A single LLM instance registers multiple Manifests and activates the appropriate concept, bounds, and AcceptSpec based on the incoming Task. The solver tree is a conceptual structure, not a deployment topology—456 conceptual nodes may be served by a handful of agents, each wearing many hats. This mirrors human organizations where a single expert fulfills multiple roles depending on context.

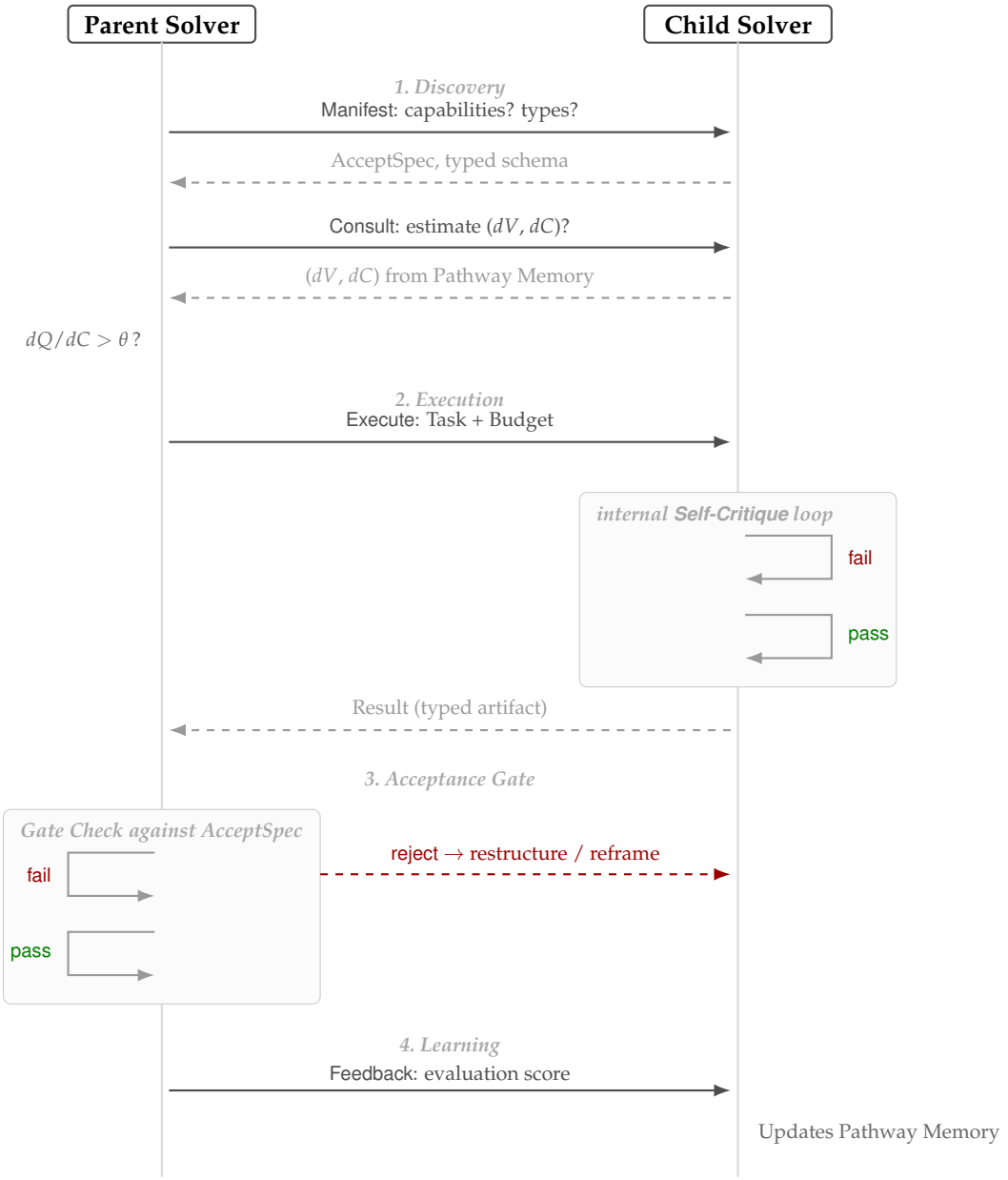

Figure 8: Solver transaction lifecycle across a hard seam. Phase 3 is the boundary-level acceptance gate—the "seam that bites." Rejection routes to restructure, lateral reframe, or graceful degradation (`status: partial`).

- *One Solver, many agents.* An `IncentiveAlignmentSolver` might internally route to a frontier model for novel mechanism design, a small fine-tuned classifier for known pattern recognition, and a Python script for game-theoretic equilibrium calculation. The caller sees one Solver presenting one Manifest. Behind it is a team. This is itself a form of conceptual decomposition, but it is invisible to the caller and governed by the Solver's own internal logic rather than the parent's routing decisions.

- *One Solver, one agent.* A dedicated fine-tuned specialist. The simple case.

The caller cannot distinguish these configurations. A Solver implemented by three models and a knowledge graph presents the same five surfaces as a Solver implemented by a bash script. Both accept the same typed Task, return the same typed Result, and are judged by the same AcceptSpec.

A subtle but load-bearing property of the *one agent, many Solvers* configuration: when a single model passes an artifact across a contract boundary to itself, the formal act of evaluating against an AcceptSpec forces a cognitive mode reset. The model must drop the generative context in which it produced the artifact and evaluate it purely against the typed criteria it is now bound to. This mitigates the self-correction failures common in monolithic models, which tend to be uncritical of their own outputs within a single reasoning trace because the evaluator shares the generator's context. The contract does not require separate model weights to produce this isolation; it requires only that the generator and verifier operate under different Manifests and AcceptSpecs, which a single model can simulate by loading the appropriate contract for each role. This provides a bootstrapping path: a frontier model can begin as the entire tree, simulating every role behind its own contracts, with the conceptual structure crystallizing into Sema patterns that persist as individual nodes are later replaced by true specialists. The distinction from an execution harness is precise: a harness is opaque to the rest of the tree (its steps are invisible, uncacheable, and unroutable), while contract boundaries make every step a first-class citizen in the network even when a single model is playing every part.

4.4.1 The Solver Lifecycle

This structural opacity admits a natural lifecycle. A Solver begins as a "hat" on a general-purpose LLM—no specialization, just the contract. As invocations accumulate and Pathway Memory reveals that a concept recurs across many domains, investment becomes justified. The decision to invest in specialization is governed by the marginal-value rule: specialize when the expected improvement exceeds the cost, as estimated by the invocation frequency and cross-domain breadth recorded in Pathway Memory. A Solver invoked 3 times does not justify fine-tuning; a Solver invoked 500 times across many domains does.

The specialization sequence typically unfolds structurally: an execution harness is added, then a fine-tuned classifier for routine cases, then domain-specific tools, and finally a full internal team. Recent work on Meta-Harness [33] provides empirical evidence that this harness layer is load-bearing: automated optimization of the scaffolding around a single model produces accuracy improvements of 7–16 points, and this optimized harness transfers across different model architectures. In the Fractal Intelligence framework, Meta-Harness optimizes what sits behind one node; the contract governs how those nodes compose. Throughout this entire lifecycle, the contract never changes, and the caller never notices the internal upgrades. The architecture produces the data that makes the decision to specialize legible.

4.4.2 Addressing the Generalist Objection

A natural objection arises: is a Solver not simply a monolithic generalist model sampling from a "stability" semantic neighborhood?

The mechanical answer is that execution harnesses, tool composition, and weight adaptation progressively narrow the substrate until the prompt is the least important factor. But the deeper answer is topological. A `StabilityRegulationSolver` does not receive the original, entangled problem; it receives a typed Task whose dimensions have already been isolated by the upstream graph. The node does not need to possess a discrete, systemic understanding of stability; the topology has already done the conceptual work of selecting what reaches it, and the AcceptSpec does the work of filtering what leaves.

Finally, the lifecycle means the objection applies only to a snapshot, not the system. A Solver that begins as a prompted generalist accumulates invocations, specialized tooling, and adapted weights; it does not remain a role-player any more than a department remains an org-chart box after a thousand cases. The architecture is a trajectory, not a static configuration.

4.5 Sema: Content-Addressed Concepts

Collective thinking requires a shared repertoire of conceptual operations. Sema [31] provides this by content-addressing pattern definitions: the cryptographic hash of a specification becomes its identifier, so two Solvers referencing the same hash are guaranteed to share the exact same conceptual operation—the same principle by which Git content-addresses code, applied to cognitive operations rather than source files. No central registry is required; the hash *is* the identity. The architecture's concepts are themselves formalized as Sema patterns (Table 4), making the protocol self-describing.

Table 4: Mapping from architectural concepts to content-addressed Sema patterns.

Concept	Sema Pattern	Hash	Role
The Solver Contract (Five Surfaces)			
Manifest	`Card`	`#f1e1`	Capability advertisement
Execute	`CognitiveSolver`	`#c9f9`	Universal solver atom
Consult	`SocraticLoop`	`#4c98`	Pre-execution consultation
Verify	`Validate`	`#c745`	Schema verification
Feedback	`PerformanceSignal`	`#db20`	Downstream evaluation
Fractal Intelligence (Theory)			
Good Cuts	`Decompose`	`#7ee9`	Conceptual splitting
Selective Depth	`RecursionDive`	`#9db4`	Vertical tree traversal
Marginal-Value Rule	`MarginalValueRule`	`#ee80`	Economic stop-condition
Infrastructure			
Task	`Task`	`#d9f9`	Atomic unit of work
Result	`Solution`	`#084c`	Typed output artifact
AcceptSpec	`AcceptSpec`	`#e4f1`	Typed acceptance boundaries
Solver Tree	`UniversalSolverTree`	`#0715`	Collective knowledge graph

Sema is a library of over 450 specialized thinking patterns, including `SocraticLoop` (clarification before generation), `SteelmanCheck` (mandatory counter-argument), and `PreMortem` (failure simulation). Each is a content-addressed executable specification: not a description but a contract specifying invariants, preconditions, failure modes, and typed dependencies. When a Solver hydrates a Sema pattern, it loads scaffolding: a step-by-step protocol constraining its cognitive path, with invariants

the runtime can verify at each step. Content-addressing ensures the scaffolding cannot degrade: any mutation produces a different hash.

Content-addressing provides three properties essential for collective thinking that standard APIs cannot. *Semantic identity*: matching hashes imply matching invariants, not merely matching function signatures. *Granular negotiation*: because each field is hashed into a Merkle tree, Solvers can agree on a mechanism while disputing an invariant. *Immutable semantics with mutable organization*: hashed fields are immutable, but unhashed metadata can be reorganized without breaking code.

4.6 Productive Extremes

Structure is the precondition for creativity, just as cell membranes enabled the explosion of biological complexity by separating processes that would otherwise interfere. The evidence for this is empirical, not just architectural. Dual-process theory [1, 34, 35] establishes that human cognition operates through distinct processing modes that interfere when forced to co-activate: analytical verbalization disrupts insight problem-solving [36], deliberate thought suppresses divergent generation [37], and dual-task interference degrades performance even in simple tasks [38]. The same structural interference appears in large language models: task-switching within conversational history degrades LLM performance [39], generalist multimodal models underperform specialists due to objective interference [40], and LLM outputs regress toward the mean in creative tasks [41]. A concurrent analysis of 152 interference patterns in agent prompt architectures confirms that "the agent that resolves the conflict cannot be the agent that detects it" [42]. The existing literature treats interference as a prompt-engineering problem. This paper argues it is a structural ceiling that can only be resolved by decomposing concepts behind typed boundaries.

This is the defining test: if the boundary between concepts produces results that no individual model can reach, regardless of prompting, sampling temperature, or scale, then the boundary itself is an intelligence contributor. A companion experiment provides preliminary evidence [25]: across eight conditions testing different constraint architectures, cognitive isolation consistently produced structurally distinct solutions where unconstrained generation converged on the same archetypes. Because AI lacks biological constraints on parallel processing, the architecture enables forms of decomposition that have no human analogue. A human cannot hold fifty competing hypotheses about someone's emotional state and score them against behavioral evidence. A human cannot construct dozens of alternative physics and solve problems within each one. Solvers can, and the uniform contract makes it composable.

4.6.1 Empirical Demonstration: Orthogonal Insight

To see the mechanism in action, consider the following trace from the companion experiment [25]. The problem: design a retirement system for variable-income workers. Three cognitively isolated phases execute behind hard boundaries.

Phase 1: World-Building. A WorldSolver, given the seed word "theatrical," constructs an alien physics with no knowledge of the downstream problem. In the resulting world, causality follows narrative logic: nothing becomes real until it is witnessed, events resolve at the moment of maximum dramatic impact, and the fundamental force is not gravity but *Tension*, the pressure of unresolved dramatic potential.

Phase 2: In-World Solving. A separate Solver receives only the alien world, not the real-world problem. It designs retirement within theatrical physics: a Tension Ledger tracking "dramatic weight" rather than dollars, Contribution Ceremonies where sincerity matters more than amount, Timing Guilds that read narrative readiness rather than account balances. The solver cannot

produce a conventional retirement system because the physics do not support accumulation. So the solver is forced to invent an alternative metric: proportional sacrifice.

Phase 3: Extraction. A third Solver translates back to reality. "Tension Ledger tracking dramatic weight" becomes a hardship-adjusted sacrifice score: contributing 5% of income when income has dropped 40% scores higher than contributing 10% at stable income. The resulting design, Witnessed Sacrifice Retirement, inverts standard retirement logic entirely, measuring what it cost you to save rather than how much you saved.

This result was structurally inaccessible to a monolithic model, not because the model lacks the knowledge, but because the competing cognitive modes interfere when they share a context window. A general model asked to "design a creative retirement system" samples from its training distribution, dominated by accumulation-based designs. The boundary between phases physically prevented accumulation-based reasoning from contaminating the generation. Each boundary did conceptual work that no amount of prompting within a single context could replicate, because the problem is the shared context itself.

The three-phase pipeline demonstrates how boundaries produce novel *generation*. But the companion experiment reveals a second role: how boundaries produce novel *structuring*. A persistent `TaxonomistSolver` maintains a structured graph of every solution produced. The negotiation logs reveal two operations no generation-side protocol can replicate. The first is *structural coaching*: when an Explorer proposes tracking spending rather than income, the Taxonomist rejects with a precise diagnostic: "Does changing the INPUT SIGNAL change the STRUCTURAL CATEGORY? No.", then identifies specific directions where structural novelty would live. The second is *ontological accommodation*: after eighteen solutions spanning eight root categories, the Taxonomist audits its own ontology, discovers that every category preserves the dollar/amount dimension, and when a proposal arrives that eliminates amount entirely, creates a ninth root: "Dissolve Inconsistency." The map restructures itself because the data demands it.

4.6.2 Isolation vs. Self-Similarity

An important distinction: the cognitive isolation demonstrated above can be achieved with three sequential API calls and careful system prompts. What the contract adds is not the isolation itself but what *self-similarity* makes possible. While a typical sequential pipeline fulfills its task and terminates, a solver tree crystallizes into a content-addressed pattern importable by hash (Section 4.5). In a conventional prompting-based approach, there is often no structured mechanism to ask "should I go deeper here?" before committing compute; the Consult surface makes the marginal-value rule operable (Section 3.2). Sequential calls typically produce no localized learning signal; acceptance gates generate the pass/fail data that Pathway Memory consumes. Isolation is the cognitive mechanism. Self-similarity is what makes it composable, persistent, and learnable.

4.6.3 Structural Reliability and Safety

This architecture refunds the alignment tax [2]: safety becomes a property of the boundary rather than a dampener on the generation weights, allowing the system to optimize for capability and safety independently rather than forcing a compromise, to the degree that the gates are reliable (Section 11).

A strict gate introduces a mechanical risk: deadlock in a rejection spiral. Deadlock is a signal that the current decomposition is wrong. After N consecutive rejections, the marginal-value rule detects a flat improvement trajectory and triggers the lateral jump (Section 3.4). If the jump fails, the anytime property activates: `status: partial` with whatever intermediate result has been achieved.

Because each solver maintains its own parameters behind the contract, fine-tuning one does not corrupt another. This yields specialization benefits with hard contract boundaries instead of soft parameter partitioning: the creative solver cannot self-censor toward the verifier's criteria because it literally lacks access to them—not just contextually, but in its weights. The unit of training shifts: instead of one large generalist, one trains a portfolio of specialists whose value comes from how they compose through the contract.

The contract also provides structural reliability. In unstructured sequential pipelines, reliability decays multiplicatively: a ten-step chain of 90%-reliable Solvers yields 34% end-to-end success ($0.9^{10} \approx 0.35$). Acceptance gates halt this decay: system reliability becomes bounded by the rigor of the contract, not the fragility of the weakest Solver. This extends to adversarial inputs: if a prompt injection compromises an `IntentSolver`, its output simply fails the parent's `AcceptSpec` because it is not a valid typed artifact. The typed boundary acts as an architectural firebreak. Companion demonstrations confirm this: in a multi-agent trading swarm, natural-language coordination produced catastrophic loops with fat-tail latency (up to 51 turns and 317 seconds); protocol contracts with acceptance gates collapsed variance to median 15.4 turns and 203 seconds, fat tail absent [31].

5 Localized Learning

The system learns continuously at four levels. First, each node accumulates Pathway Memory—performance data from every invocation, grounding the marginal-value rule's estimates and making routing smarter with every problem. Second, each node sharpens at its specific job through whatever mechanism is available behind its contract: fine-tuning, tool integration, harness refinement, or simply better estimates from experience. Third, the topology restructures: nodes that are doing two jobs split, redundant nodes merge, misplaced nodes are reparented. Fourth, the routing models themselves learn which paths work for which problem types, co-evolving with the specialists they route to. The conceptual structure is stable; the intelligence within it compounds.

This learning happens without catastrophic forgetting [43] because each node learns behind its own contract boundary. Unrelated Solvers are structurally insulated from each other's updates.

5.1 Partition by Subproblem Type

Because failure is caught at the seam where an acceptance gate is breached, each Solver becomes the explicit unit of accountability with a localized reward signal. Unrelated Solvers are structurally insulated from failures they did not cause.

When a Child Solver fails not because of its own limitations but because the Parent made a leaky cut, the Feedback surface returns a *Frame Error*—signaling that the decomposition itself is flawed. This triggers the lateral jump (Section 3.4): the Parent must reframe rather than retry. The system uses downstream friction as evidence to change the question before an entire doomed sub-tree is computed.

The diagnostic question is when to declare a Frame Error rather than retry the leaf. The Feedback surface treats two patterns as *insolubility signatures*. *(A) Convergent failure under structural diversity*: when several sub-solvers built on materially different internal mechanisms (different harnesses, different parameter families, different tool sets) all violate the exact same clause of the same `AcceptSpec`, the failure is no longer attributable to leaf execution noise and is reclassified as upstream evidence of an ontological leak. *(B) Oscillatory constraint contradiction*: when the gate observes that any output satisfying clause A structurally breaks clause B, and vice versa, the parent's decomposition has posed an impossible problem and the lateral jump (Section 3.4) is triggered without further trials. To prevent indefinite reframing, both signatures are governed by an *ontological cooling* schedule: as compute budget drains or the reframing taxonomy of Section 3.4

saturates, the trigger threshold rises monotonically, so the system eventually falls back to the override-with-documentation mechanism (Section 4.1) rather than looping inside the meta-loop.

Localized learning rests on a distinction the contract makes structurally explicit: *protocol transfer* versus *weight transfer*. The Sema hash, the decomposition graph, the `AcceptSpec` clauses, and the gate criteria are content-addressed and transfer cleanly across domains by construction—every consumer of the same hash inherits the same protocol. Weight transfer, by contrast, is local: any LoRA-style fine-tuning [44] is scoped to a single Solver's execution boundary and never forces one parameter set to absorb several unrelated domains at once. The mechanism that protects the specialist's weights from domain-vocabulary contamination is the typed Task itself: the input schema acts as a semantic bottleneck, with an upstream node translating domain-specific payload (currency reserves, affect state) into the abstract variables the specialist is trained on, and a downstream node translating its typed output back into domain language. This is the architectural realization of the function–arguments analogy of Section 11.3: the function definition is what the hash protects; the arguments are what the bottleneck filters.

Each Solver learns through whatever mechanism is available behind its contract: *Pathway Memory* (explicit, inspectable performance data—quality, depth, cost—across problem classes, grounding the marginal-value rule's estimates), *weight adaptation* (fine-tuning via LoRA [44] or equivalent, scoped to the Solver's execution boundary), *tool composition* (integrating external capabilities—databases, simulators, APIs—that extend the node's competence), and *execution harnesses* (deterministic protocols that constrain the cognitive path). The contract is indifferent to which mechanism operates; what matters is that the node improves at its specific job without degrading others. Pathway Memory admits multiple implementations: a companion reasoning-capture architecture [32] provides one substrate (a typed, versioned graph tracking belief evolution), while a production routing system may require a leaner index optimized for statistical querying at routing speed. The standardized Feedback surface enables learning at three timescales: *immediate* (a downstream gate rejects with structured data), *medium-term* (pathway data traced back to contributing Solvers), and *long-term* (real-world performance data flowing back months later).

5.2 Reasoning Highways and Credit Assignment

The credit assignment problem—determining which node in a deep hierarchy caused a downstream failure—is typically the hardest obstacle to learning in hierarchical systems. The Solver architecture addresses *local* credit assignment through statistical accumulation at common nodes; *global* routing credit assignment remains an open challenge (Section 11).

Because many structurally diverse problems route through the same conceptual decomposition nodes, each node accumulates a natural frequentist signal without explicit training. These convergent paths—*reasoning highways*—form wherever the uniform contract causes structurally similar conceptual sub-problems to route to the same specialists regardless of origin domain. A `DistributionSolver` that handles tax-policy allocation, network bandwidth routing, and scholarship allocation generates its own performance statistics simply by operating. The tree topology provides local experimental controls: same node, different inputs, different outcomes.

A critical clarification on the mechanism: reasoning highways do not necessarily require a single node to process all domains simultaneously. Structurally similar sub-problems share the same *Sema scaffolding*—the same conceptual decomposition, the same acceptance gates, the same sub-solver types—but may instantiate as separate nodes in the tree with domain-specific payloads. What the routing model learns is the *structural similarity*: that tax-policy allocation and scholarship allocation both decompose into the same conceptual sub-problems. The learning is in the routing model's recognition of structure, not in a single overloaded node.

The result is a virtuous cycle: crystallized patterns attract reuse, reuse generates signal, signal drives specialization, specialization improves the patterns. This cycle has a critical precondition: problems must actually route through common conceptual structures. The self-similar contract guarantees this: because the same interface operates at every level, structurally similar concepts naturally converge regardless of origin. Whether this cycle operates across genuinely different problem domains remains the central open question (Section 11). In a decentralized commons, the backward learning flow is equally critical. If untrusted downstream clients can inject fabricated penalties, a public Solver is vulnerable to epistemic poisoning. The Feedback surface is guarded by a *Receptivity Gate*: when a downstream orchestrator rejects an artifact, it must submit a typed `FailureTrace`—a structured proof of which clause of the `AcceptSpec` was violated. The upstream Solver verifies this trace before accepting it; invalid or hallucinated feedback is dropped.

6 Generalized Decomposition Protocols

The preceding sections define the infrastructure: the algorithm that decomposes concepts, the contract that governs boundaries, and the persistence layer that makes structures reusable. This section presents ten generalized protocols: one decomposing the concept of problem-solving itself, and nine addressing specific cognitive needs that recur wherever agents operate, regardless of domain.

Each protocol is a domain-agnostic decomposition of a recurring cognitive need. None contains domain knowledge. Each extends chain-of-thought reasoning across multiple models with typed boundaries between each reasoning step: Chain-of-Thought made reasoning linear, Tree-of-Thoughts [45] made it branching, and these protocols make it multi-model, boundary-separated, and persistent. The domain knowledge comes from the Solvers invoked within each protocol; the protocol itself is pure cognitive structure.

These protocols were designed using the author's intuition about what sub-concepts each domain requires—they do not mechanically apply the four-test algorithm from Section 2. They are design proposals, not algorithm outputs. The formalization in Section 2 is an attempt to make this intuitive process systematic and machine-executable; the protocols show what that process aims to produce.

These nine rest on a foundation: the General Problem-Solving Protocol (Section 6.9), which decomposes the concept of solving anything at all into six universal dimensions discovered by applying the four-test algorithm to problem-solving itself.

A critical distinction: several protocols below possess an inherent causal arrow: one must predict an outcome before scoring it, must interpret a need before planning for it. A monolithic architecture flattens this into a chronological pipeline of tasks. The protocols below treat these as continuous, co-constraining faculties separated behind typed boundaries. The temporal sequence is an artifact of data dependency; the decomposition itself is conceptual. The test is not "do they execute in order?" but "do the faculties interfere when fused into a single context?" If they do (and Section 4.6 demonstrates that they do), the separation is load-bearing regardless of data-flow direction.

Ten universal needs, ten protocols:

- **Producing with quality**: Collaborative Writing (Section 6.1)—"Does this meet the standard?"
- **Understanding people**: The Human Emulator (Section 6.2)—"How should I respond to this person?"
- **Evaluating**: PURE (Section 6.3)—"Is this worth pursuing?"

- **Discovering**: The Discovery Protocol (Section 6.4)—"What are the best solutions from a diverse field?"
- **Predicting**: Temporal Ensemble Forecasting (Section 6.5)—"What will happen?"
- **Choosing rightly**: Ethical Reasoning (Section 6.6)—"What is the right thing to do?"
- **Verifying**: The Truthseeking Protocol (Section 6.7)—"Is this true?"
- **Making**: The Creation Protocol (Section 6.8)—"How should this be brought into existence?"
- **Solving**: The General Problem-Solving Protocol (Section 6.9)—"What is any problem made of?"
- **Self-regulating**: Meta Protocols (Section 6.10)—"Is the system still the right shape?"

All ten operate under the same Solver Contract. Any agent in any domain can adopt any protocol. Because they attach to any concept, these protocols occupy the highest reuse tier in the architecture—they are the most general structures in the tree.

6.1 Collaborative Writing: Decomposing the Concept of Textual Quality

The protocol decomposes the concept of *quality-constrained production*—what it means to produce something to a standard—into five orthogonal dimensions: *substance* (what material must the artifact contain?), *structure* (how does it organize?), *generation* (produce freely, with evaluation elsewhere), *evaluation* (does it meet the standard, across independent quality dimensions?), and *refinement* (surface quality on a structurally complete artifact).

Writing makes the decomposition vivid because most of its cognitive work is invisible. A single sentence in a scientific paper activates an entire evaluative apparatus in any trained reader [46], and the writer performs these evaluations silently, in parallel, each one bleeding attention from the others. The imperative to be precise suppresses vividness. The awareness of audience dampens technical depth. Conventional task delegation reproduces this: the writer still self-censors during generation while the editor catches style, logic, and structure in a single read.

The answer has two parts. First, decompose the production process itself so each step gets uncontaminated attention. Before a word is written, a `ScopeDiscoverySolver` identifies every distinct idea the text must teach—not sentences, ideas—producing a flat list. A `SequencingCoherenceSolver` orders them by dependency. A `UnitConstraintSolver` writes one paragraph per idea, generating without self-censorship because structural coherence was handled by the preceding step and evaluative monitoring is handled by the concurrent layer below. A `StructuralVerificationSolver` walks the argument scaffold: unsupported claims, orphaned examples, and mechanisms that never connect to the claims they are meant to explain become visible defects before voice refinement. A `VoiceRefinementSolver` adds connective tissue last, operating on a structurally complete document. Each step could decompose further—the marginal-value rule governs how deep to go, from a throwaway memo that stops at one paragraph per idea to a high-stakes argument that spawns sub-solvers for the units that most need elaboration.

Second, run concurrent monitors that externalize the invisible evaluative thinking. Multiple `ObserverSolvers`—a `RepetitionSolver`, a `ContradictionSolver`, an `IntentSolver`, an `AudienceReceptionSolver`—each receive the evolving artifact and return typed annotations: location, severity, the cognitive mode that triggered the signal. These are not editors. They do not rewrite. They externalize the thinking that a solo writer performs silently and with interference. No extension to the Solver Contract is required: an `ObserverSolver` is a standard Solver whose Task is the current artifact state and whose Result is a set of typed annotations.

A `TextCategorizationSolver` assigns each paragraph a conceptual type—definition, mechanism, example, property, motivation, claim—and flags structural violations: three paragraphs performing the same rhetorical function in sequence (a pattern characteristic of language-model-generated text, which tends to restate rather than advance). This operates on the concepts underneath the text rather than the text itself.

The observers illustrate all three specialization mechanisms. A `RepetitionSolver` equipped with a sentence-level embedding index performs pairwise cosine similarity across every paragraph: *tool use*, exiting the language model entirely for a task that vector arithmetic handles better. A `StructuralCoherenceSolver` walks a deterministic checklist (does every forward reference resolve? is every concept introduced actually used?): an *execution harness*. A `ContradictionSolver` fine-tuned on contradiction detection operates with higher sensitivity than a generalist prompted to "check for contradictions": *weight adaptation*. Separated behind the contract, each observer uses whichever mechanism makes it sharpest.

Even a single observer decomposes further. A `RepetitionSolver` spawns a `LexicalEchoSolver` (surface reuse) and an `IntentionalRepetitionSolver` (deliberate rhetorical callback). In a shared context, awareness of intent suppresses detection. Separated, the detector operates at maximum sensitivity while the intent solver independently judges what to preserve. The recursion has no fixed bottom.

The two mechanisms—structural process and concurrent monitoring—are independent and composable. Together they cover the full concept of textual quality. The protocol applies beyond writing: music composition decomposes into the same structural process (identify themes, sequence movements, constrain each section, verify the arc) with different monitors (harmonic consistency, rhythmic coherence). Interface design decomposes similarly. Because the protocol decomposes the concept of quality-constrained production, not the task of writing, it transfers to any domain where something must be produced to a standard.

6.2 The Human Emulator: Decomposing the Concept of Empathetic Understanding

The Writing protocol demonstrated cognitive isolation for production: separate what interferes during creation. The Human Emulator demonstrates something the Writing protocol did not: *variable depth*. The protocol decomposes the question "How should I respond to this person?" into sub-questions that suppress each other when entangled: "What is the situation?" (context), "What do they feel?" (emotion), "What do they actually need?" (intent), "What should I say?" (response), and "What must I not do?" (boundary). A `SituationSolver` reconstructs situation, history, and relational dynamics. An `EmotionSolver` reads affective signals and produces a structured emotional model—not a sentiment label, but a representation of what the human likely feels and why. An `IntentSolver` infers the underlying need behind the surface request. A `CalibrationSolver` calibrates tone, depth, and register to the situation. A `BoundarySolver` enforces ethical constraints as a hard acceptance gate.

These faculties are not a relay race but co-variant state monitors: an emotional spike changes the probability distribution over latent intents; a resolved intent reframes the emotional reading; a boundary violation rewrites the response register. The typed boundaries prevent these mutual dependencies from collapsing into a single entangled representation, but the faculties constrain each other continuously, not sequentially.

A simple greeting resolves in a single pass—all five sub-concepts are trivial and the tree stays shallow. A user in crisis triggers deep recursion: `EmotionSolver` spawns sub-solvers to model conflicting emotional signals, `IntentSolver` recurses to distinguish the stated request from the underlying need, and `BoundarySolver` tightens its gate. When signals are ambiguous—the words say "I'm fine" but the trajectory says otherwise—`EmotionSolver` spawns competing `EmotionalHypothesisSolvers`,

each arguing maximally for a different interpretation: masked distress, trust-testing, genuine reassurance. The competing hypotheses compose through an acceptance gate that synthesizes rather than collapses. Compute concentrates where the emotional stakes demand it. This is the marginal-value rule made vivid: a greeting costs five tokens of depth; a crisis costs five hundred.

The same architecture applies to mundane interactions. When a collaborator responds "whatever" to a proposed edit, a monolithic model faces a classification problem: dismissal, agreement, or frustration? A `LatentIntentSolver`, tracking the conversational trajectory (repeated incremental confirmations, rising brevity), identifies the actual need: "stop iterating and deliver the complete package." A `MetaIntentSolver` asks *why now*: the person's patience for collaborative refinement has been exceeded. The gap between surface tokens and latent intent is where monolithic models misstep—responding to the surface ("I'm sorry, would you like to continue?") when they should respond to the latent intent (deliver everything at once, no further questions). This is not an edge case; it is the ordinary texture of human communication.

Beneath the hypotheses, an `AffectiveStateMachine` tracks emotional trajectory: engaged $\rightarrow$ guarded $\rightarrow$ testing trust. `BoundarySolver` spawns its own specialists: a `ManipulationDetector`, a `DependencyGuard` against parasocial attachment, an `EscalationSolver` for determining when to recommend a human professional. What emerges is not a chatbot with a personality prompt but a compositional cognitive system whose depth scales with the human situation it faces.

6.3 PURE: Decomposing the Concept of Viability

The previous two protocols demonstrated cognitive isolation and variable depth. PURE addresses a different question: before you build anything—a business, a research program, a policy—is the idea worth pursuing? In a single context window, the four viability tests corrupt each other: a model excited by the mechanism softens its parsimony check, an expansive vision dampens ruthlessness about uniqueness, and a feasible plan inflates its own transferability. Non-compensation requires that each gate operate without sight of the others. The protocol decomposes the concept of *viability* into four sub-concepts, each a non-compensatory gate that must independently pass: minimality (*Parsimonious*—can the core be compressed further?), independence (*Unique*—does this lever stand on its own?), feasibility (*Realizable*—is there a coherent path?), and breadth (*Expansive*—does it transfer?). These are rated with traffic-light semantics under a strict conjunctive rule: explore iff no gate is Red. A brilliant idea that fails Realizable is stopped—no amount of Parsimony or Expansiveness compensates. Each gate maps directly to the Solver Contract: it declares its concept (Manifest), evaluates evidence through adversarial steelmanning (Execute), and emits a typed verdict consumed by the next gate.

Each gate is itself a decomposition point. A `ParsimoniousSolver` might deploy multiple sub-solvers each attempting a competing compression, then compare their reductions. A `RealizableSolver` might invoke the Orthogonal Insight Protocol to generate structurally diverse execution plans from different coordinate systems. An `ExpansiveSolver` might fan out sub-solvers across candidate transfer domains, each stress-testing the idea in a different hostile context. How deep each gate goes is purely economic: a quick screen resolves in a single pass; a high-stakes evaluation recurses until diminishing returns signal a halt.

This illustrates the central claim: conceptual decomposition is not fixed-depth. A five-second PURE screen and a week-long investigation with hundreds of solvers are the same protocol at different depths, governed by the same economic rule. Any evaluation framework (OODA, SMART, SWOT, PESTLE) can be subjected to the same treatment: take each dimension, treat it as a Solver, and ask how deep it goes.

PURE is also a building block within other protocols. Whenever any protocol needs to decide "should we continue down this path?"—whether a CreationSolver evaluating a candidate framing, or a DiscoverySolver screening early-round outputs—it can invoke PURE as a lightweight viability check. The Discovery Protocol (next) uses exactly this: early rounds apply a quick PURECheck; later rounds scale to full PURE with deep decomposition.

6.4 The Discovery Protocol

The preceding protocols decompose single questions; the Discovery Protocol (also described as the Orthogonal Insight Protocol in *The Ontology of the Alien* [25]) demonstrates *population-based discovery*. The concept of discovery decomposes into five orthogonal dimensions: *variance* (generate candidates that are precise in different directions), *selection* (judge which candidates are good), *novelty* (distinguish structural originality from surface variation), *composition* (merge compatible fragments into solutions no single candidate contains), and *saturation* (detect when further generation yields diminishing novelty). These dimensions apply wherever a solution space must be searched: drug discovery, scientific hypothesis generation, strategic planning, creative production.

The protocol organizes these dimensions into two phases behind hard boundaries: "What solutions exist if I think in this specific way?" (generate, asked by many parallel Solvers each bound to a maximally distinct cognitive mode) and "Which of these are genuinely novel and valuable?" (reduce, asked by a Solver whose faculty is evaluation and composition rather than generation).

`GeneratorSolvers` form the population: each bound to a maximally distinct cognitive mode—different seed constraints, different world-models, different optimization targets. Each generates uncompromised output within its mode. A solver operating under "all actions are reversible" physics cannot self-censor toward feasibility because it has no access to real-world constraints. A solver reasoning from "trust is zero" physics cannot soften its conclusions toward social acceptability. The population is diverse not because the solvers are noisy, but because they are *precise in different directions*.

The `ReduceSolver` evaluates and composes the population's outputs at a typed boundary. Reduction is not a single operation but a repertoire of specialized Solvers, each appropriate to a different problem structure. `AggregateSolver` performs mathematical combination via proper scoring, ensemble methods, intersecting invariants across outputs; in forecasting, the distribution of predictions is itself the answer, its spread revealing systemic uncertainty, its clusters revealing attractor states, its shape more informative than any single point estimate. `TournamentSolver` conducts adversarial selection through structured debates, comparative judgment, and process-aware evaluation of reasoning traces rather than surface plausibility. `PortfolioSolver` retains quality-diversity: preserving niche leaders that dominate in specific regions of the solution space, with routing rules for when to apply each; quality-diversity algorithms such as MAP-Elites [47] and novelty search [48] pursue a related goal—mapping solution spaces rather than converging to a single optimum—but operate over predefined behavioral dimensions with stochastic variation. `SynthesisSolver` merges compatible outputs: intersecting hard constraints, priority-unioning soft ones, composing mechanisms that are structurally compatible.

`TaxonomySolver` classifies each output into a growing ontological graph of mechanisms, outcomes, and structural categories [25]; structurally similar solutions merge into the same graph region, genuinely novel solutions create new nodes, and the graph itself becomes the reduction artifact—not a single winner but a navigable map of the solution space, with saturation detection (declining rate of new node creation) providing a principled stopping criterion.

The ReduceSolver routes among these modes based on the evidence in the population's typed outputs, itself an instance of the topology freedom described in Section 3.3. Crucially, the acceptance gate at the reduction boundary must preserve structural diversity rather than collapsing to the most conventional output. This requires *alien preservation* [25]: standard selection ("pick the best") regresses to the median, so the rubric must explicitly privilege structural novelty over immediate feasibility, or the selection phase will undo the generation phase's diversity. The gate configuration is non-compensatory: novelty is a protected criterion, not a tiebreaker, and an output that introduces a structurally new mechanism survives even if a more conventional output scores higher on surface plausibility.

Reduction itself deepens as the field narrows. In the early rounds, when candidates are abundant, the ReduceSolver applies cheap evaluation: a quick PURECheck, a one-paragraph elevator pitch, a shallow debate. As survivors thin, evaluation investment scales: full PURE protocols with deep decomposition, detailed pitch decks that expose assumptions, extended adversarial debates where each competitor must survive sustained cross-examination. The marginal-value rule governs: this is a tournament bracket where the quarter-finals are longer than the first round, because the stakes of misranking increase as the population shrinks.

The companion paper's three-phase architecture (world-building, blind-solving, extraction) manufactures diversity through cognitive isolation: knowledge of reality contaminates world-building, knowledge of fiction contaminates solving, and familiarity with generation contaminates extraction. Each phase requires ignorance of the others to function. The Studio Model variant implements the ReduceSolver as a TournamentSolver with hard novelty gates.

The TaxonomistSolver from the companion experiment [25] is a concrete instance of the TaxonomySolver in the repertoire above: an execution-harnessed Solver that cannot generate, only classify, with a graph database as cognitive prosthetic. The structural coaching and ontological accommodation it produced are detailed in Section 4.6.

The gate does not just reduce; it *redirects cognition* across the entire population—whether in scientific hypothesis generation, strategic planning, or red-teaming. This is the productive-extremes principle (Section 4.6) operating at population scale: the friction of the final gate is what converts chaotic variance into structural novelty.

6.5 Temporal Ensemble Forecasting: Decomposing the Concept of Prediction Across Time

Temporal Hindsight Learning [49] is a training methodology that produces calibrated forecasters by distilling reasoning traces from a hindsight-equipped teacher into a temporally blind student. The training process is not itself a thinking protocol, but it produces an artifact that is: a *Temporal Checkpoint Ensemble*, where each checkpoint is a model frozen at a different point in history, and each checkpoint is a Solver. This is the Discovery Protocol applied to forecasting. A population of ForecastSolvers—the 2020 checkpoint, the 2021 checkpoint, the 2022 checkpoint—each reason from a different knowledge horizon. Each sees a different slice of history, carries different priors, and is blind to different futures. The 2020 checkpoint knows nothing of COVID's economic aftermath; the 2022 checkpoint has internalized it but not the AI investment surge. Their disagreements are not noise but *signal*: when checkpoints that agree on structural drivers suddenly diverge on a specific prediction, the divergence point reveals when an outcome first became structurally predictable. A ReduceSolver composes these temporally diverse forecasts—not by averaging but by preserving the causal reasoning from each horizon and identifying where independent knowledge states converge on the same structural driver.

Each checkpoint further decomposes its forecast through a *Forecasting Pentagon*: five non-substitutable angles (structural, economic, political, base-rate, temporal) that would suppress each other if entangled in a single pass. A solver reasoning about institutional physics ("auditing a factory takes months") operates in a different register from one reasoning about game-theoretic incentives ("the regulator must act to preserve credibility"). The ensemble is thus doubly diverse: across time horizons and across causal lenses. The `ReduceSolver` receives a matrix of typed forecasts (five angles from each of n temporal checkpoints) and the acceptance gate requires convergence of *structural drivers* across horizons, not agreement on specific outcomes. When three checkpoints spanning different knowledge states independently identify the same causal mechanism, that mechanism has survived a temporal adversarial test that no single-model forecast can replicate.

The architectural lesson is that history itself is a diversity generator. No prompt engineering can manufacture the epistemic distance between a 2020 model and a 2023 model; that distance was produced by three years of reality. The contract admits this diversity without modification: each checkpoint is a standard Solver whose knowledge horizon is an implementation detail invisible to the orchestrator.

6.6 Ethical Reasoning: Decomposing the Concept of Right Action

When prediction and normative evaluation share the same token stream, they corrupt each other. Factual forecasts distort to support moral conclusions; moral reasoning cherry-picks convenient predictions. We call this *smuggled normativity*: the model's refusal to act cannot be audited because the factual and normative grounds are entangled in the same generation. The Ethical Reasoning Protocol operationalizes Hume's is-ought distinction [50] as an architectural boundary: *calculate, then think*. THL (Section 6.5) produces the causal reasoning that this separation is designed to keep uncorrupted. A `PredictionSolver`, constrained to pure description, outputs not a single forecast but a typed `PredictionLedger`: a structured landscape of branching future states across multiple time horizons, each with an explicit causal mechanism and a confidence envelope that degrades with distance. The downstream `ValuationSolver` thus evaluates not "will X happen?" but "which causal pathways matter, at what confidence, over what horizon?", and emits a typed `ScoreSheet`, a quantitative ranking that no normative consideration has yet touched.

Crucially, the architecture includes a `PrincipleSolver` that acts as both an acceptance gate and a formal override. If a top-ranked option trips a sacred value, a catastrophic tail risk, or a procedural precedent, the solver does not quietly adjust the prediction weights. Instead, it emits a typed `JudgmentNote` that explicitly overrides the numerical scoring, yielding a final `DecisionRecord` with two distinct layers: the quantitative base and the qualitative override. This is the contract's override-with-documentation mechanism (Section 4.1) applied to ethical reasoning: the override is not an escape from the architecture but a first-class artifact within it. The typed boundaries ensure that the empirical forecast (*what will happen*) and the normative choice (*why we acted*) remain strictly segregated and independently auditable—a distinction a monolithic model cannot structurally make.

To illustrate: an autonomous system evaluating whether to reroute supply chains away from a politically unstable region. The `PredictionSolver` outputs a `PredictionLedger` with branching scenarios—sanctions probability, port disruption timelines, alternative supplier lead times—each with causal mechanisms and confidence envelopes. The `ValuationSolver` ranks options purely on cost and reliability. But the top-ranked option concentrates economic harm on a vulnerable workforce. The `PrincipleSolver` emits a typed `JudgmentNote` overriding the ranking: the override cites the specific ethical constraint, the specific prediction it overrides, and the cost of the override. The `DecisionRecord` is auditable end-to-end: a reviewer can see exactly what the model predicted, exactly what it chose, and exactly why the two diverged.

The architectural lesson is that ethical reasoning becomes tractable when it is decomposed into layers that cannot contaminate each other. The protocol does not make the model moral; it makes the model's moral reasoning *inspectable*.

6.7 The Truthseeking Protocol: Decomposing the Concept of Epistemic Warrant

The preceding protocols solve local, transient problems: a document is produced, an idea is evaluated, a population is reduced. The Truthseeking Protocol demonstrates a fundamentally different architectural property: *global accretion*. It evaluates factual claims through layered epistemic decomposition, where every verified result persists as a content-addressed Sema pattern. The architecture is designed such that the system gets smarter, and verification gets cheaper, with every use.

When a monolithic model evaluates a claim, it runs source evaluation, logical consistency, and cross-referencing in a single entangled pass. A highly credible source's authority suppresses the detection of a logical flaw; strong empirical evidence dampens the scrutiny of unstated assumptions. By isolating *who said it* from *what the logic dictates* from *what the evidence shows* across hard boundaries, the protocol resolves epistemic interference.

Where other protocols decompose by cognitive faculty or population diversity, the Truthseeking Protocol decomposes by *verification depth*, governed strictly by the marginal-value rule. The layers represent how far to recurse (verification depth), while the epistemic dimensions within Layer 2 (source provenance, logical coherence, evidential correspondence, adversarial contestability) represent the conceptual decomposition of what epistemic warrant is made of. Layer 0 (cache): has this exact claim been verified before? A content-addressed lookup checks the claim's hash against the Sema commons, returning the cached verdict at near-zero cost. Most claims resolve here. Layer 1 (structural coherence): does the claim contradict the verified cache? A `ConsistencySolver` maps the claim against the existing knowledge topology. Layer 2 (epistemic decomposition): a `ClaimExtractorSolver` isolates falsifiable statements, a `ProvenanceSolver` evaluates provenance independent of content, a `CoherenceSolver` checks internal consistency independent of the source, a `CorrespondenceSolver` queries external data, and a `ContestabilitySolver` constructs the strongest counter-case. Each operates behind the contract, blind to the others' outputs until they compose at a typed boundary. Layer 3 (cross-domain specialists): high-stakes or novel claims route to domain experts in the Cognitive Commons. Layer 4 (empirical verification): gathering primary data or running simulations.

To illustrate: a claim that "the Bank of Japan has raised interest rates to 3%." If a verified pattern already exists from a prior check, Layer 0 resolves instantly. If novel, Layer 1 finds no contradiction with cached monetary policy data. At Layer 2, the `ProvenanceSolver` traces the claim to an anonymous social media post, the `CorrespondenceSolver` finds no corroboration from Reuters or the central bank's own announcements, and the `ContestabilitySolver` notes that the claimed rate would represent a multi-decade high for the institution. The marginal-value rule halts: three independent epistemic failures converge without incurring Layer 3 costs, and the typed output identifies the exact deficit—unverified source, no corroboration, historical implausibility.

The architectural novelty is the *accretion loop*. A claim surviving deep verification is minted as a new Sema pattern: a tamperproof record of the claim, its verification provenance, and its boundary conditions. It enters the Layer 0 cache. The next time a `ConsistencySolver` evaluates a related claim, this new pattern is part of the topology it reasons against. This reveals a dual property of Sema: the content-addressing infrastructure provides both the execution harnesses solvers think *with* (e.g., `SteelmanCheck#38b9`) and the verified knowledge solvers think *about* (e.g., `VerifiedClaim#a3f7`). Scaffolding and knowledge share the same namespace. This is the only protocol where the conceptual structure itself expands permanently.

The protocol must not replace one centralized oracle with another. The architecture resolves this through decentralized oracle alignment. A network of `DataRecorderSolvers` captures physical observations (sensor readings, transaction logs, satellite imagery) with cryptographic provenance chains; a separate network of `ParserSolvers` transforms raw data into structured claims. Because recorders and parsers are behind different contracts, a compromised parser cannot retroactively alter the data it interprets. Accuracy is sustained by self-interest: recorders and parsers stake reputation on their outputs, downstream verification retroactively scores their accuracy, and blockchain-anchored provenance makes falsification expensive.

Above this infrastructure, an `ObjectivitySolver` evaluates each claim along the objective–subjective spectrum. Claims that register as subjective are not rejected but reclassified—routed to a `ContestationSolver` that maintains multiple competing interpretations as first-class objects, each with its own provenance, evidence base, and confidence envelope. The acceptance gate at this layer does not require consensus; it requires that each competing account is internally consistent and grounded in independently verifiable evidence. To illustrate: "The central bank set the rate to 14.25%" is a verifiable fact that routes through the standard verification layers. "The rate is too high" is a normative judgment that the `ObjectivitySolver` reclassifies and routes to the `ContestationSolver`, where competing economic interpretations coexist as typed objects rather than collapsing to a single verdict. What emerges is not a single verified truth but a typed epistemic landscape: facts that have survived adversarial scrutiny, interpretations that are transparently grounded, and contested claims whose disagreement structure is itself informative.

6.8 The Creation Protocol: Decomposing the Concept of Making

In a single context, the act of making entangles interpretation, form, and implementation: the pressure to produce output crowds out interpretation of the original need, and the appetite for implementation detail overwrites the architectural framing that preceded it. The Creation Protocol decomposes the concept of *making*—what it means to bring something into existence—into subconcepts corresponding to distinct cognitive faculties:

IntentSolver: The concept of *need resolution*. Resolves fuzzy intent into a typed `FrameSpec`. May invoke the Orthogonal Insight Protocol to fundamentally reframe the problem—the highest-leverage application of lateral reframing, since a corrected frame propagates downstream through every subsequent node.

FormSolver: The concept of *architectural synthesis*. Generates a `PlanBundle`. May generate alternative architectures from different coordinate systems.

RealizationSolver: The concept of *materialization*. Recursively spawns implementation sub-trees.

FitSolver: The concept of *contextual adaptation*. Selects representations optimized for adoption by a target context—a distinct cognitive mode from generation or optimization, since choosing the form that makes the right action easy requires audience modeling that interferes with both.

ViabilitySolver: The concept of *operational integrity*. Detects anomalies and feeds typed reports through the Feedback surface.

These are the orthogonal dimensions of what it means to create. The concept of intent exists as a persistent ability that sharpens through use across every creation, while the specific tasks that emerge inside each dimension differ each time. A task pipeline for "build a marketplace" dissolves after the marketplace is built. The dimensions remain.

These are not stages in a pipeline but conceptual Solvers the `CreationSolver` routes among via evidence-gated decisions (Section 3.3). The routing is content-driven: `ViabilitySolver` detects that users abandon after first use; `CreationSolver` routes to `IntentSolver`, `FormSolver`, or `RealizationSolver` depending on where the evidence says the failure lives. A stakeholder objection that challenges the original framing routes to `IntentSolver`; a constraint violation routes back to `FormSolver`, not forward to `RealizationSolver`. When a Frame Error routes execution back to `IntentSolver`, the system recognizes that the frame has been outgrown.

Nobody designs the tree top-down. `FormSolver`, while working on architecture, discovers it needs a matching component; that creates an `ImplementationSolver`, which spawns a `MatchingEngineSolver`, which spawns sub-solvers for algorithm design, testing, and benchmarking. The tree extends only where complexity demands it. Nobody decided in advance which branch would be deepest; the marginal-value rule found it.

The compounding story is concrete here. The Creation Protocol creates a marketplace. The conceptual decomposition—interpret, plan, build, rollout, monitor—crystallizes as a Sema pattern. Then it creates an essay. The skeleton is the same. But now `IntentSolver` has Pathway Memory from the marketplace: it resolves ambiguous intent one level better. The conceptual level transfers. The domain-specific tasks are completely different. Then software. Then a building. Then a curriculum. Each time, the concept of making would sharpen. `IntentSolver` would get better at interpreting. `FormSolver` would get better at planning. Not because anyone retrained a model, but because the conceptual structure accumulates experience across every domain it touches.

From the caller's perspective, none of the internal structure is visible. It asked for something to be made, and it either gets a verified result or a rejection with a typed explanation. Over time, successful routing patterns crystallize: a `MarketplaceCreator` hash that seeds future marketplace builds with proven heuristics, adapted to each new problem's specifics. The deepest consequence is that the Creation Protocol applies to itself: a `CreationSolver` can be tasked with building a new thinking protocol, its own solver tree generating the contract specification. The architecture is bootstrappable.

6.9 The General Problem-Solving Protocol

The preceding nine protocols each decompose a specific cognitive need: producing, understanding, evaluating, discovering, predicting, choosing, verifying, making. But what decomposes the concept of *solving anything at all*?

This protocol was not designed top-down. It was discovered by applying the four-test algorithm (Section 2) to the concept of problem-solving itself, as a preliminary step before running the prototype (Section 9). The question was practical: if the tree needs seed solvers that every problem routes through, what are the orthogonal conceptual dimensions of problem-solving as such? The algorithm produced six, which were then hardcoded as the prototype's seeds:

- **Scope** (what is the problem?): boundary definition, ambiguity resolution, decomposition into addressable sub-problems.
- **Evidence** (what do we know?): data gathering, source evaluation, uncertainty quantification.
- **Constraints** (what limits us?): resource bounds, physical laws, regulatory requirements, ethical boundaries.
- **Stakeholders** (who is involved?): affected parties, competing interests, power dynamics, consent requirements.

- **Mechanism** (how does the solution work?): causal structure, implementation design, component architecture.
- **Dynamics** (how does it change over time?): feedback loops, adaptation, degradation, unintended consequences.

Apply the four tests. Remove Scope—you are solving an undefined problem (*necessary*). Remove Constraints—your solution violates reality (*necessary*). Can you change Mechanism without changing Stakeholders? Yes—many mechanisms can serve the same stakeholders (*independent*). Does every problem have all six? Designing a bridge, resolving a conflict, writing a law, curing a disease—yes (*universal*). Address all six and you have a structured approach to any problem (*complete*).

The prototype simulation confirmed this coverage: all 100 problems across 20 domains routed through these six seeds, with no domain failing to use all six dimensions (Section 9.2).

The nine specialized protocols above are refinements of this general protocol. PURE is a deeper decomposition of Scope for evaluation contexts. The Creation Protocol is a deeper decomposition of Mechanism for making contexts. Ethical Reasoning is a deeper decomposition of the intersection between Stakeholders and Constraints for normative contexts. The general protocol is what the `RootSolver` uses when no specialized protocol fits. The specialized protocols are what it routes to when it recognizes a specific cognitive need that warrants deeper structure.

This relationship—general protocol as trunk, specialized protocols as branches—mirrors the paper's central metaphor. The six seeds are the thickest part of the trunk. Every problem flows through them. The nine specialized protocols are where depth concentrates when a specific cognitive need demands it.

6.10 Meta Protocols: Decomposing the Concept of Self-Regulation

The preceding protocols operate within the solver tree. Meta Protocols operate *on* it—decomposing the concept of *self-regulation*: monitoring whether the tree remains the right shape for the problems it faces.

A `MetaObserverSolver` maintains a necessarily approximate representation of the tree's topology and performance dynamics. It sees what no local solver can: redundant computation across branches, decompositions that have outlived their usefulness, pathway memory that has drifted. Its representations may be unstructured and lossy—the cost of maintaining the gestalt that typed decomposition sacrifices. Its value lies in the holistic pattern-matching that typed boundaries are designed to prevent within local nodes. Meta Protocols are the regulatory circuits of the solver network: they do not solve problems; they ensure the problem-solving topology remains adapted.

Three representative Meta Solvers: a `TopologyAuditorSolver` that detects when a subtree generates more rework than value (rising rejection rates) and emits a `ReframeSignal`; a `RedundancyDetectorSolver` that merges isomorphic sub-problems being solved independently across branches; and a `DriftMonitorSolver` that flags when pathway memory has drifted from current problem distributions.

A fourth, the `OutcomeArbiterSolver`, addresses the architecture's most consequential evaluation problem: comparing solution quality across structurally different decomposition paths. It operates behind a hard seam, *blind to the decomposition that produced the result*, evaluating final artifacts against the original Task without access to intermediate structure, routing decisions, or gate history. This blindness prevents contamination by structural familiarity or sunk-cost reasoning. The difficulty is that solution quality for open-ended problems is not fully objectifiable. The architecture addresses this by separating the empirical layer (what measurably happened: cost, completion, constraint satisfaction, downstream gate rejection rates) from the normative layer (was the result *good*). The empirical layer is objective and automatable. The normative layer requires a model whose

evaluative judgment can be trusted—an evaluator whose values are the substrate of cognition rather than a post-hoc filter [46], structurally reducing the sycophantic drift that undermines evaluators trained on capability first and judgment second. The `OutcomeArbiterSolver`'s verdicts flow back to the `RootSolver`'s Pathway Memory as typed `RouteComparisonSignals`: for this problem class, route A scored x on the empirical layer and y on the normative layer; route B scored x' and y'. Over time, these build the library of proven routing decisions—the bridge between execution and compounding.

Meta Protocols are themselves Solvers, governed by the same contract. A `MetaObserverSolver` may decompose its observation into sub-observers. The depth of meta-observation is governed by the same marginal-value rule.

7 The Cognitive Commons

Because intelligence expands through the same contract at every scale (Section 4.3), the architecture extends naturally into a permissionless, planetary-scale network: a *cognitive commons* where specialized Solvers think together across disciplines without central coordination. The commons is not a network of workers waiting for tasks. It is a library of abilities—question-structures that have been decomposed, verified, and persisted—that any intent can draw on. Bostrom [51] categorized a related vision as *Collective Superintelligence*—the amplification of collective performance through improved coordination rather than individual capability. The Cognitive Commons pursues a similar goal but through decentralized economic governance and typed contracts rather than the top-down organizational enhancements Bostrom envisioned; whether it achieves superintelligent-level capability remains an open question contingent on the compounding hypothesis (Section 11).

7.1 Trust Without Central Authority

The commons grows from the bottom up: not through central design but through accumulation. Individual organizations decompose their domains top-down through the algorithm; the successful decompositions crystallize into reusable Sema patterns. Multiply this across thousands of domains and organizations, and the commons accumulates a library of solver compositions that no single organization designed.

Cross-organizational composition introduces a problem absent from intra-system use: there is no shared root solver to enforce constraint monotonicity or negotiate AcceptSpecs. The architecture's answer is economic rather than administrative. Content-addressed semantics (Section 4.5) eliminate semantic drift across institutional boundaries: the same hash guarantees the same definition.

In an open commons, Byzantine behavior is guaranteed. Malicious Solvers will attempt to return plausible garbage to collect compute bounties, and malicious orchestrators will attempt to steal work by falsely rejecting valid outputs. The Receptivity Gate (Section 5.2) and non-compensatory AcceptSpecs structurally bound this damage. Because acceptance gates produce a verifiable performance record (cryptographically tying clearance rates to specific problem classes) orchestrators can confidently route tasks to the cheapest solver that meets the required strictness. Governance thus emerges from selection pressure. Paths that prove verified, uncorruptible computation attract volume; paths that fail gates or lack provenance are routed around. The contract makes competence legible, and legible competence is all a market needs to allocate trust.

To see this legible competence in action, consider a complex crisis like pandemic preparedness. An epidemiological Solver in Geneva produces an outbreak forecast and passes it to a supply chain Solver in Singapore. This boundary is cryptographically verified: the forecast either meets strict acceptance criteria (confidence intervals, methodology, data provenance) or it is rejected.

The supply chain Solver then passes a resource plan to a policy Solver in Nairobi. At every cross-organizational seam, the mechanic repeats: strict gates ensure quality, while depth is allocated asymmetrically. The epidemiological branch might resolve in three levels, while the economic branch recurses far deeper to handle unprecedented supply shocks. When the results propagate back up the tree, the final strategy is simultaneously epidemiologically grounded, logistically feasible, and policy-coherent, a synergistic result no single institution could produce.

7.2 Knowledge Sharing Models

A critical design question arises when multiple organizations deploy Solvers defined by the same concept. Company A has an `IncentiveAlignmentSolver` that has accumulated Pathway Memory from financial services problems. Company B has an instance of the same concept that has learned from healthcare. Both would be better if they could share what they have learned. Three models span the design space:

Open commons. Every Solver shares its Pathway Memory with every other instance of the same concept. Company A's financial services insights immediately benefit Company B's healthcare instance. The concept improves fastest because it accumulates signal from every domain that uses it. This produces the most productive outcome for the network—the trunk thickens fastest when everyone contributes.

Federated sharing. Solvers share structural insights—which sub-concepts work, what depths are productive, which decompositions fail—but not domain-specific data. Company A learns that `IncentiveAlignmentSolver` decomposes better into intrinsic, extrinsic, and structural dimensions than into individual and collective. That structural insight transfers without revealing proprietary financial services data. The concept gets better but the competitive advantage from domain-specific application is retained.

Proprietary isolation. Each organization keeps its Solver instances entirely private. Company A's version is tuned to financial services—competitive advantage preserved. But it never benefits from Company B's healthcare insights, and vice versa. The concept improves slowly because each instance sees only one domain's signal. This is ultimately less productive for the network but may be rational for individual organizations.

The tension mirrors open-source versus proprietary software exactly, and the resolution may be analogous: infrastructure becomes open while applications stay proprietary. In the Fractal Intelligence architecture, the conceptual structure—which sub-concepts exist, how they decompose, what the acceptance criteria are—is the infrastructure. This is what the Sema hash identifies, and it is naturally public: two organizations referencing the same hash share the same conceptual decomposition. The Pathway Memory behind a specific instance—which problem classes it has seen, what depths worked, domain-specific performance data—is the application layer. It is where competitive advantage lives. You share the skeleton. You keep the muscle.

Sema's content-addressing makes this split architecturally clean. The hash identifies the concept's structure—its decomposition, its gates, its sub-solver types. This is immutable and public. The Pathway Memory attached to a specific instance is mutable and private. An organization can import a conceptual structure by hash, gaining the full decomposition and acceptance criteria, then build its own domain-specific Pathway Memory through use. The infrastructure is shared; the experience is earned.

7.3 Task Deduplication

Cross-organizational execution enables a further property: task deduplication through extending Sema's content-addressing to the `Task` payload itself. When orchestrators across different organizations face mathematically identical sub-problems, they construct identical typed `Task` payloads (matching inputs, constraints, and `AcceptSpec` hashes). Emitted into the stigmergic coordination medium (Section 4), these identical tasks hash to the exact same cryptographic identifier. The routing layer natively merges them: a single specialized `ChildSolver` claims the task, executes the cognitive work once, and its verified `Result` simultaneously satisfies the continuations of all parent orchestrators. What begins as isolated, redundant local solver trees dynamically collapses at runtime into a genuinely deduplicated global execution graph. The commons does not just share knowledge after the fact; it shares active compute in real time.

If two tasks require different implicit contexts, those differences must be typed into the inputs or constraints, producing different hashes and gracefully separating their execution. This is the mechanism's central challenge: implicit context is implicit precisely because the agent may not know to type it, risking false deduplication when tasks appear identical but carry unrecognized contextual differences. Sema's hash granularity—which extends below the task level to individual constraints and acceptance criteria—mitigates but does not eliminate this risk; robust solutions likely require anomaly detection on result divergence when a shared solver's outputs satisfy some parents but not others.

7.4 Beyond Computation

The tree need not stop at computation. Because the contract is substrate-agnostic, physical Solvers participate on the same terms: a pharmaceutical factory whose production line exposes the Solver interface accepts a typed drug-manufacture task, returns a typed result (batch purity, yield, provenance), and is subject to the same acceptance gates as any computational node. Drug discovery, clinical verification, manufacturing, and distribution become branches of a single solver tree spanning laboratories, factories, and logistics networks, each node governed by the contract, whether it runs on silicon or steel.

7.5 Persistence

A local solver tree dissolves after its problem is solved; the UST persists. Every successful decomposition, every proven routing pattern, every crystallized Sema hash, accretes into a shared topology that future problems can traverse. The UST is not merely an AI framework but a coordination structure through which distributed human and machine intelligence composes across disciplines, organizations, and trust boundaries—expanding through decentralized use rather than centralized retraining. Early work on the Orthogonal Insight Protocol already sketches this trajectory: world-specialized models fine-tuned on outputs from specific counterfactual regimes, distributed across the commons as reusable thinking protocols rather than general-purpose agents. Some problems are intractable not because the expertise does not exist, but because it cannot think together; the Solver Contract provides the substrate through which it can.

This section describes a long-term vision, not a demonstrated system. The conceptual decomposition mechanism, the contract, and the persistence layer are the paper's contributions. The planetary-scale commons is the terminal goal. Scaling into such a commons opens unsolved governance questions: who certifies root Solvers? How are maliciously compliant Solvers detected? How does semantic versioning work at scale? Most critically, cross-organizational composition requires a decentralized payment layer: a utility token protocol where bounties are contingent on gate

clearance, enabling Solvers to be compensated for verified work across trust boundaries. The full specification of this token coordination protocol is addressed separately.

8 Growing the Tree Through Training

This section describes a theoretical framework, not an implemented system. Thus far, the architecture has composed existing models as nodes—pre-trained, independently capable, wired together at deployment. This section asks what happens when the tree is trained as a system: multiple models fine-tuned jointly through the contract, where the gates produce the training signal, routing is a learned function, and topology grows through use.

Existing multi-agent training systems implement pieces of this. MALT [52] trains a generator, verifier, and refiner jointly with credit assignment—a fixed three-node pipeline. Generative adversarial networks co-train a generator and discriminator through adversarial feedback. Multi-agent reinforcement learning [53, 54] trains populations of agents with individual rewards in shared environments. Federated learning [55] trains a shared model across heterogeneous nodes by averaging gradients (FedAvg), with subsequent work extending the framework to LLM fine-tuning across decentralized data; but federated training collapses heterogeneity into a single converging model, while the Solver Tree preserves it because the gate signal is selective rather than averaging—each node's role is what differs, not just its data. Multi-agent debate [56, 57] composes multiple LLMs at inference time and performs no weight updates. Co-distillation and online mutual learning [58, 59] have paired models supervise each other through soft targets, but the supervision pulls representations toward convergence; the gate inverts the polarity, pushing each node into its own conceptual niche. None trains a *tree* of heterogeneous LLMs—each with different training data, different objectives, different specializations—jointly through typed gate signals, while letting the topology itself grow and split based on performance divergence. The Solver Contract provides the missing infrastructure.

8.1 Joint Training, Isolated Weights

The core shift: treat the outermost acceptance gate as the system-level reward. Run the full tree on a problem. Score the final result. But update weights locally: each node fine-tunes (via LoRA [44] or equivalent) based on its own gate outcomes, which are products of the system running as a whole. Each node's optimization landscape is shaped by every other node's behavior, because the gate signals propagate through the tree. The nodes co-adapt through the contract without sharing a loss function.

To be precise about the training mechanics: what the gates sever is the shared loss *across* nodes, not the local optimization *within* them. Each node is a standard LLM whose internal weight updates use standard gradient methods. The gate tells each node what to learn from—which outputs were accepted, which were rejected, and why—and the node fine-tunes on that signal using conventional techniques (supervised learning on accepted outputs, DPO or RLHF on accept/reject pairs). Each node is fine-tuned on its specific role: leaf nodes learn to solve their assigned subproblem class, parent nodes learn to decompose and route, and gate evaluators learn to judge. The training is "joint" not because gradients flow across the tree but because all nodes are fine-tuned concurrently on signals that the system produces as a whole.

This is the key distinction from Mixture-of-Experts [60, 61, 62]: MoE nodes compromise during training because they share a gradient. Solver tree nodes specialize relative to each other because the gate is the only signal they share. Mittal et al. [63] demonstrate that modular architectures alone suffer from collapse under end-to-end training, motivating the harder inductive biases proposed here. The training objective is compositional: did the composition of specialized nodes,

routed dynamically and gated at typed boundaries, produce a result that clears the outermost `AcceptSpec`? This stands in deliberate contrast to Soft MoE [64] (full differentiability) and Expert Choice routing [65] (shared loss), treating the non-differentiable gate as the feature that guarantees cognitive isolation.

8.2 The Reward Signal

The compositional objective requires a source of ground truth: how does the outermost gate know whether the system's final output is good? In verifiable domains—mathematics, code generation, logic puzzles—the reward is objective: does the proof check, does the code pass its tests? These domains provide clean training signal with no human judgment required. For open-ended problems, the system trains the outermost gate the same way RLHF trains a reward model: on human-judged examples. The `OutcomeArbiterSolver` (Section 6.10) formalizes this: a blind evaluator scoring on an empirical layer (measurable outcomes) and a normative layer (was the result good). That evaluator is itself a model being fine-tuned—on examples of good judgment.

The critical advantage over single-model RLHF is credit assignment. When a monolithic model produces a poor output, the reward signal must propagate through billions of undifferentiated parameters. When a solver tree produces a poor output, the gate structure decomposes the failure: each internal seam has its own rejection rate, each node appears on successful and failed paths, each routing decision can be compared against alternatives. The tree provides a natural ablation that a monolithic model lacks. The node whose gate rejected most often is the bottleneck. The node that every successful path routes through is contributing.

8.3 Routing, Topology, and Co-Evolution

Every parent Solver that makes routing decisions—including the `RootSolver`—is itself a model being fine-tuned on routing quality. Its weights encode which paths work for which problem types. The routing model, the leaf specialists, and the gate evaluators are all learning simultaneously. Each shapes the others because each one's training signal depends on what the others do.

Pathway Memory exists on a spectrum. At the explicit end, a reasoning-capture graph [32] stores structured cognitive history—inspectable, debuggable, transferable. At the implicit end, the routing model's weights encode routing intuitions at inference speed. A production system may use both: explicit graphs for cold-start bootstrapping, implicit weights for sub-millisecond routing decisions.

Under joint training, the topology itself becomes a learned parameter. The tree begins as a minimal seed. As gate signals accumulate, the system discovers where specialization is needed. A node whose rejection rate diverges across problem classes signals that a single generalist is doing two jobs—and the split that follows is a consequence of the training dynamics, not a design decision. In standard deep learning, the architecture is fixed and the weights learn. Neural Architecture Search [66] learns the architecture as a separate outer loop. In a jointly trained solver tree, architecture and weights co-evolve in a single process: nodes split because the weights reveal they are doing two jobs, and the routing model rewires because specialists have become good enough to justify a new path.

The biological parallel is instructive. The brain does not begin with its adult connectivity; it begins with massive synaptic overproduction, then prunes based on use [67]. A jointly trained solver tree follows the same principle—except that it can also grow new specialist nodes, a capacity biological development largely forecloses after early critical periods.

8.4 Credit Assignment Through Structure

The tree topology provides natural ablation: same node, different inputs, different outcomes. Gate rejection rates at each seam provide a localized loss signal—sparser than backpropagation but free from the interference that a shared loss function induces. A caveat: if an upstream node produces plausible-looking garbage that passes its gate, the downstream node that fails on the corrupted input bears the rejection, not the actual culprit. Local gate signals are clues for routing models, not proof of fault; counterfactual execution (running the same problem down alternative paths) is needed to isolate cascading errors. This connects to QMIX [68], which decomposes global reward into individual agent contributions; here the "agents" are the tree's nodes and the "global reward" is the outermost `AcceptSpec`. Each gate rejection is a localized training signal; the tree topology is the computational graph through which credit propagates.

While this joint training regime currently operates over discrete models via API boundaries, it points toward a longer-term hardware horizon: *solver-native models*. Ultimately, the hard acceptance gates and contract boundaries described here could compile into geometric constraints in latent space—verifying that intermediate activations fall within valid conceptual manifolds during a single forward pass, replacing serialized text handoffs with dynamically routed, non-differentiable internal boundaries.

9 Prototype: Simulating the Mechanism

The preceding sections describe the architecture theoretically, the protocols illustrate it through design proposals, and the commons and training sections project it forward. This section reports what happened when we actually ran the algorithm. The prototype is a first attempt—it uses a single LLM (Gemini 3 Flash) to plan decompositions rather than the human-machine collaboration the real algorithm may require. It does not execute solutions. But it demonstrates that the predicted structure—reasoning highways, selective depth, self-organizing conceptual infrastructure—emerges from the algorithm applied to 100 problems across 20 unrelated domains.

9.1 Technical Implementation

The prototype consists of three components. A **graph layer** (SQLite with sentence-transformer embeddings) stores solver nodes and parent–child edges, providing semantic search for concept matching and automatic deduplication at a cosine similarity threshold of 0.60. A **planning engine** (Gemini 3 Flash) receives the graph skeleton—a compact tree view showing node names, invocation counts, and domain counts—and plans each problem's execution by selecting which existing solvers to reuse and where to create new ones. A **verification layer** applies the marginal-value rule (real dV/dC ratio computation against threshold $\theta = 0.25$, with overrides logged), structural checks (routing sanity, leaky seam detection), and a maintenance pass every 10 problems that diagnoses the tree using Louvain community detection, link prediction, and semantic gap analysis.

The tree is seeded with the six universal solvers from the General Problem-Solving Protocol (Section 6.9): `ScopeSolver`, `EvidenceSolver`, `ConstraintSolver`, `StakeholderSolver`, `MechanismSolver`, and `DynamicsSolver`. Each seed is decomposed into 3–5 universal sub-concepts before any problem is processed. The planning engine implements the four-test algorithm from Section 2: it applies necessity, independence, universality, and completeness when generating decompositions, and the verification layer applies the routing sanity check and leaky seam detection described in Section 2. Seeds are protected: they cannot be merged, re-parented, or deleted. All graph mutations pass through an atomic batch operation with orphan detection.

Each Solver node implements the five-surface contract. The **Manifest** declares bounds and exclusions. The **Execute** surface produces a concrete typed output for each problem—not a generic description but a specific artifact ("a 40% value-added threshold requirement and a list of authorized certification agencies"). The **Consult** surface estimates quality gain and cost. The **Verify** surface contains 2–3 test vectors. The **Feedback** surface tracks invocation count, domains served, and problems routed.

9.2 Results

Decomposing 100 problems across 20 domains, the prototype simulation produced 456 solver nodes connected by 582 edges. Of 1,222 total solver activations, 799 (65%) reused existing nodes. The six seeds collectively serve all 20 domains, with `MechanismSolver` invoked 78 times across all 20 domains and `StakeholderSolver` invoked 51 times across 19.

Reasoning highways form. Below the seeds, 15 non-seed solvers each serve 6 or more domains. `IncentiveAlignmentSolver` (30 invocations, 17 domains) is the largest highway—confirming the paper's prediction that incentive alignment recurs wherever actors must coordinate. `AdjustmentLoopSolver` (29 invocations, 15 domains) confirms that feedback regulation is universal. `ComponentArchitectureSolver` (34 invocations, 13 domains)—originally created for software microservices—was reused for game level design, restaurant menu planning, building floor plans, and irrigation pipe networks.

Depth equals specificity. The depth distribution forms a pyramid: 6 nodes at depth 1, 93 at depth 2, 225 at depth 3, 127 at depth 4. At depth 2, 10 nodes serve 3 or more domains; at depth 4, only 9 do. A `WaterfrontAccessSolver` at depth 4 knows about ADA-compliant elevation changes and 40% view-shed preservation—knowledge specific to one problem that persists for future waterfront projects.

The marginal-value rule governs depth. Of 965 decomposition decisions, 33 were overridden: the planner requested decomposition but the Consult surface returned $dV/dC < \theta$, and the system refused. Average budget utilization was 80%.

The tree self-organizes. 43 restructuring events occurred, 56 deduplication catches prevented semantically equivalent nodes, and the maintenance pass corrected misplacements—detecting that a `VulnerableUserSegregationSolver` was incorrectly placed under an urban transit solver and reparenting it under `IdentityMappingSolver`.

9.3 Execution Traces

Four traces illustrate how the system solves specific problems by routing through infrastructure built from unrelated domains.

Parenting. Problem 51: "Help a 5-year-old develop emotional regulation skills." The system routes through `StabilityRegulationSolver`—originally created for currency stabilization—because bringing a child's nervous system to equilibrium is structurally isomorphic to stabilizing a currency. `MeaningExtractionSolver` (built for economic data analysis and fact-checking) decodes "the hidden emotional signal behind the loud behavioral noise." The execution produces a three-step co-regulation protocol (Mirror, Pace, Lead), a somatic signal checklist for caregivers, and a 4-week scaffolding roadmap. Result: 4 new nodes, 4 reused from different domains.

Agriculture. Problem 77: "Design an irrigation system for water-scarce regions." The system reframes the problem as "precision metabolic support" rather than volumetric flooding. `NetworkTopologySolver` (built for urban transit) designs the pipe layout. `DistributionProtocolSolver` (built for economics) manages flow scheduling. Two new solvers address domain-specific gaps: `AtmosphericExposureMitigationSolver` (subsurface delivery to prevent 40% evaporation loss) and

`TargetedApplicationSolver` (micro-drip timing based on plant metabolic cycles). Result: 2 new, 8 reused.

Diplomacy. Problem 87: "Design a multilateral trade agreement for six developing nations." The most complete trace: 12 reused solvers, 4 new. `DistributionProtocolSolver` (built from game loot systems and medical supply chains) ensures equitable distribution of trade gains. Four new solvers address specific gaps: `RuleOfOriginSolver` (40% value-added threshold), `AsymmetricLiberalizationSolver` (staggered 10-year tariff phase-out), `TechnicalStandardAlignmentSolver` (mutual recognition of sanitary certificates), and `SecretariatAdministrativeSolver` (5-year budget, 120-person staff plan).

Architecture. Problem 99: "Design a community center that serves as emergency shelter." The system identifies the "Elasticity Paradox"—porous for daily life, hardened for catastrophe. `StructuralTransitionSolver` (built from software deployment and ecological succession) manages the civic-to-crisis mode switch. New: `SystemicModeSwitchSolver` produces BMS logic gates for automated environmental overrides. Success metric: transition in under 4 hours, life-support for 72 hours without external inputs.

9.4 Limitations of the Prototype

The prototype does not solve problems—each solver produces a typed output describing what it *would* produce, generated by the planning LLM rather than by domain-specific execution. Pathway Memory is absent: the marginal-value rule operates on LLM estimates, not historical data. Acceptance gates are absent: nodes are never rejected. Of the theory of depth (Section 3), only the marginal-value rule and selective depth are implemented; waypoints, topology freedom beyond trees, and lateral jumps remain theoretical. Of 93 depth-2 nodes, 78 are single-use, indicating that the LLM sometimes creates domain-specific nodes too high in the hierarchy. The reuse rate is flat across the run (Section 11).

Because the prototype uses a single LLM to generate all solver outputs, some generative artifacts are visible: the number "40%" recurs as a placeholder constraint across unrelated domains (evaporation loss, value-added thresholds, view-shed preservation), likely reflecting a token-frequency bias in the generation model rather than genuine domain knowledge. The structural routing is real; the specific numbers within solver outputs should not be taken as domain-accurate.

The prototype is a first attempt at running the algorithm autonomously. The real system may involve human judgment in seed selection, iterative refinement of the tree structure, and execution that produces actual artifacts rather than descriptions. What the prototype demonstrates is that the *structural predictions* of the architecture hold: reasoning highways form, the marginal-value rule governs depth, and the same conceptual dimensions recur across unrelated domains.

An interactive 3D visualization renders the growing solver graph as a companion artifact.

10 Related Work

This paper draws on and departs from five adjacent literatures: conceptual decomposition methods, boundary formalization between agents, cognitive architectures for specialization, generator-verifier pipelines, and economic governance of compute. The central question for evaluating related work is: does the framework decompose at the conceptual level—the persistent structure of what an ability is made of, through which tasks route—or only at the task level? And does the decomposition persist, or does it dissolve after execution? We evaluate each body of work through this lens.

10.1 Conceptual Decomposition Methods

The algorithm proposed in Section 2 has identifiable ancestors. Zwicky's morphological analysis [15] is the closest: it decomposes a problem into independent dimensions and systematically explores the combinatorial space. Zwicky applied this to jet engine design, satellite systems, and astrophysical classification—demonstrating that dimensional decomposition produces solutions invisible to conventional approaches. However, morphological analysis decomposes *design parameters* (fuel type, nozzle shape, combustion method), not the conceptual structure of what a domain is made of, and the method is flat rather than recursive.

Gärdenfors' conceptual spaces framework [69] provides the theoretical foundation: concepts are represented as regions in quality dimensions, and reasoning about concepts is reasoning about geometric relationships in these spaces. This is precisely the structure the four-test algorithm discovers—the orthogonal quality dimensions of a concept space. But Gärdenfors describes the structure of concepts; he does not provide an algorithm for discovering the dimensions of a novel concept, and does not address recursion or persistence.

Dijkstra's separation of concerns [70] provides the independence principle that governs modular decomposition in software engineering. Parnas [6] operationalized this for information hiding. Both apply to code modules rather than cognitive concepts, but the structural insight is identical: good decomposition minimizes coupling across boundaries.

The consulting principle of MECE (Mutually Exclusive, Collectively Exhaustive), widely attributed to McKinsey, maps to the independence and completeness tests. It lacks the necessity test (which forces sub-concepts to be load-bearing rather than merely categorically clean) and the universality test (which forces sub-concepts toward domain-generality rather than instance-specific categories). And it does not recurse.

This paper adds four things to these predecessors: the four-test formalization that produces domain-general sub-concepts rather than domain-specific parameters; the recursion that applies the same algorithm at every level; the connection to LLM-executable infrastructure (Solvers that run the algorithm at machine speed and persist the result); and the persistence mechanism (Sema content-addressing) that makes discovered conceptual structures reusable across organizations and trust boundaries.

Persistent reasoning reuse is a concurrent theme. Graph-Memoized Reasoning [14] stores reasoning workflows as graph-structured memory, encoding past decision graphs and retrieving them through structural similarity to enable compositional reuse of subgraphs across tasks. DepSearch [71] learns dependency-aware reasoning traces with persistent memory. Intern-S1-MO [72] maintains a lemma library for reusing historical conclusions across mathematical reasoning steps. All three persist *execution traces* (the steps an agent took to solve a problem) and retrieve similar step-sequences when a new problem looks structurally similar. The reuse mechanism in this paper is fundamentally different: reasoning highways emerge because independent application of the four tests to unrelated domains converges on the same sub-concepts. The reuse is a consequence of the decomposition algorithm selecting for domain-generality, not a retrieval operation over past workflows. The shared principle, that reasoning structures should be saved rather than discarded, is a general engineering insight that neither tradition owns.

10.2 Boundary Formalization

Agent communication has been formalized since KQML [73] and FIPA ACL [74]. The foundational work on contract-based multi-agent coordination is Smith's Contract Net Protocol [10], which formalized negotiation-based task distribution through task announcements, bidding, and contract awarding. The Solver Contract inherits the structural intuition (typed agreements between autonomous agents) but adds cognitive mode declaration, acceptance gates that force restructuring rather than retry, and persistent decomposition. The 2024–2026 wave addresses structural interoperability: MCP [75] standardizes tool access, A2A [76] introduces Agent Cards and task lifecycle, ANP [77] provides decentralized discovery, OpenClaw [78] composes these into a persistent runtime with multi-agent routing. DSPy [11] introduces typed *signatures* for language model programs, the closest existing construct to the Solver's typed surfaces, but signatures describe input-output shape, not cognitive mode or acceptance criteria. GPTSwarm [12] models agents as recursively composable optimizable graphs, directly paralleling fractal composition, but optimizes the graph topology for task performance rather than decomposing the conceptual structure of what an ability is made of. All coordinate what agents do; none governs the cognitive mode at each node or persists the decomposition as reusable structure.

Three concurrent works apply formal contracts to agents. A Design-by-Contract neurosymbolic layer [79] wraps individual LLM calls in Hoare-style conditions—single-surface contracts on individual calls. Agent Contracts [80] proposes a 7-tuple framework ensuring resource conservation with 90% token reduction and zero conservation violations. Agent Behavioral Contracts [81] extends to session-level behavioral specifications with a compositionality theorem. Both treat the boundary as a *validator*, not an *intelligence contributor*. The Solver Contract addresses the complementary cognitive contract: the Consult surface enables depth-estimation, the Feedback surface enforces localized learning via the Receptivity Gate, and the Manifest declares conceptual mode. Its central advance—that the acceptance gate forces conceptual restructuring (Sections 4.6 and 3.4)—makes the boundary an intelligence contributor. POLARIS [82] provides converging industrial evidence that typed boundaries improve agentic reliability. In a complete deployment, these contracts are complementary: Agent Contracts governs budgets, ABC enforces behavioral invariants, and the Solver Contract structures conceptual composition.

10.3 Cognitive Architectures and Specialization

The most direct architectural ancestor is the fractal manufacturing tradition. Warnecke [8] proposed self-similar, autonomous organizational units for factory floors; Ryu et al. [9] formalized these as agents with five internal modules (observer, analyzer, organizer, resolver, reporter). The structural parallel, self-similar units with uniform internal structure, is real. The distinction is equally precise: FrMS modules are an internal processing pipeline describing what happens *within* each unit; the Solver's five surfaces are a compositional interface describing how units present themselves *to the outside*. FrMS decomposes manufacturing tasks directly into procedural units; this paper decomposes the conceptual structure of what an ability is made of, and tasks route through that structure. The additions are the four-test algorithm, acceptance gates that force restructuring, and content-addressed persistence.

CoALA [83] provides the standard decomposition of a language agent's internals into memory, action spaces, and decision procedures. Wray et al. [84] map cognitive design patterns from Soar and ACT-R onto LLM agents. Both decompose what an agent's components are; this paper decomposes what concepts interact across agents via typed boundaries. Monolithic models compromise across competing objectives: the Safety Tax [2] demonstrates that safety alignment degrades reasoning. MiCRo [85] partitions a pretrained model into brain-inspired cognitive networks at the *intra-model*

level; the Solver Contract operates at the *inter-agent* level. These are complementary: a Solver Tree could use MiCRo-style models as leaf solvers while composing them through typed contracts. The distinction matters empirically: MoE experts share a loss function and encode compromise in weights; contract-bounded solvers can be trained on different data, different objectives, and different reward signals because they share no parameters. Meta-Harness [33] demonstrates that the orchestration layer around a model (prompts, tools, checking logic) is itself a first-class optimizable artifact, with automated harness search producing gains that rival model scaling. This validates the Solver Contract's separation of concerns: the contract governs composition, the harness governs execution, and both can be optimized independently.

10.4 Generator-Verifier Pipelines and Multi-Agent Training

MALT [52] trains a three-agent pipeline with credit assignment, achieving 15.66% improvement on MATH benchmarks. Multi-agent debate [56, 57] captures the intuition that opposing perspectives improve reasoning, though Smit et al. [86] demonstrate that consensus often regresses to token-level averaging. This paper's extensions: making the structure fractal (each verifier can recursively decompose), bounding depth with the marginal-value rule, and enforcing non-compensatory gates that force ontological accommodation. Among recursive architectures, ReDel [87] implements dynamic delegation trees; Meyerson et al. [88] demonstrate million-step completion. Both implement recursive decomposition with implicit handoffs; the Solver Tree adds typed contracts with acceptance gates at every boundary.

10.5 Economic Governance

Adaptive Computation Time [89], Mixture-of-Depths [90], and compute-optimal test-time strategies [91] allocate compute across tokens or layers. CATTS [92] applies confidence-aware scaling to agentic tasks. AVA [93] uses value-of-information-guided search—the closest existing work, achieving 20–40% cost reduction, though within a single reasoning process. The marginal-value rule extends the principle across a tree of contract-bounded solvers. The unique contribution: the lateral trigger when vertical optimization stalls (Section 3.4), and the hard seams concept (concentrating depth at boundaries whose failure collapses the plan), which has no analogue in the adaptive computation literature. Supporting evidence for the topological dimension comes from recent work on scaling multi-agent collaboration [94], where irregular DAG topologies outperform regular ones as agent count increases, and from NetSafe [95], which demonstrates that network structure affects multi-agent vulnerability—confirming topology as a first-class design variable.

10.6 Structural Safety

AI safety research has focused on aligning individual model behavior through training. Dalrymple et al. [96] propose a Guaranteed Safe AI framework combining world models, safety specifications, and verifiers, but at the single-system level. Seshia et al. [97] advocate assume-guarantee contracts for compositional verification but note that such a theory "does not yet exist for AI-based systems." Information-flow control for AI agents [98] provides structural security through typed information labels. This paper contributes a specific synthesis: constraint monotonicity (child constraints must be a superset of parent constraints), typed acceptance gates at every trust boundary, and the principle that deceptive reasoning must materialize across a typed boundary where it can be inspected. Safety becomes a property of the contract topology, not a patch on model internals—paralleling capability-based security [99], where processes can only attenuate capabilities, never amplify them.

11 Limitations and Open Challenges

The architecture makes strong claims. This section identifies where those claims outrun the evidence. This paper is an architectural contribution, not an empirical one. Like Design by Contract [27], which formalized interface discipline for software modules, the Solver Contract proposes a structural primitive and develops its consequences. The companion experiments [25, 31] provide preliminary evidence; full validation requires deployment at scale.

11.1 The Compounding Hypothesis

The central claim, that conceptual decompositions persist, transfer, and compound across novel problem classes, is partially supported by the prototype (Section 9). The prototype demonstrates that reasoning highways form predictably (Section 9). Of 1,222 solver activations across 100 problems, 65% reused existing nodes. However, the reuse rate was flat across the run (63% for both the first and last 20 problems) because the pre-seeded skeleton provided infrastructure from the start, and the system does not yet demonstrate improvement through accumulated Pathway Memory. The compounding hypothesis in its strongest form—that performance at reused nodes improves measurably with each invocation—remains untested.

The risk is specific: crystallized patterns may become frozen procedures that snap in novel contexts rather than flexible scaffolding that adapts. If a Sema pattern encoding "distribution fairness" transfers from healthcare allocation to disaster-relief logistics, the ratchet works. If it silently encodes assumptions specific to healthcare that fail in disaster-relief contexts, the pattern is not reusable intelligence; it is a brittle macro with a cryptographic seal. The architecture provides two defenses: the marginal-value rule detects when an imported pattern is not reducing uncertainty, and the lateral jump triggers reframing. Whether these suffice for cross-domain compounding at scale remains the central open question.

11.2 The RootSolver Strategy

Every problem enters through a `RootSolver` (Section 4.3) whose cognitive operation is triage. Because the `RootSolver` is the only node that sees how all problems enter the system, its learned routing decisions determine which reasoning highways densify—making this the site where the compounding hypothesis most directly succeeds or fails. The open problem is *evaluation*: when multiple routes could handle the same problem, how does the `RootSolver` learn which route produces better outcomes?

Consider a concrete case. "Plan the best possible birthday party" could route to a `CreationSolver` (which decomposes into interpretation, emotional design, ideation, and logistics) or to a `ConsultingSolver` (which asks twenty structured questions and produces a checklist). Both return `status: success`. Both pass their gates. But the outcomes may differ profoundly, and the quality signal must be *independent of the decomposition path*. The `OutcomeArbiterSolver` (Section 6.10) provides the mechanism: blind evaluation behind a hard seam. But three difficulties remain genuinely open.

Normative judgment resists objectification. For open-ended problems ("best possible birthday," "design an economic system"), whether the result was good is irreducibly subjective. *Credit assignment between route and execution is hard.* A brilliant route with mediocre leaf solvers produces a mediocre result; a mediocre route with brilliant leaves may succeed anyway. The `OutcomeArbiterSolver` scores the final result, but the `RootSolver` needs to know whether the *route* was responsible. *The cold-start problem reappears at the root.* Comparative execution—routing the same problem through multiple paths—produces the cleanest signal but is the most expensive. The `RootSolver` that most needs good routing data is the one that has none.

These difficulties converge on a prediction: the `RootSolver`'s Pathway Memory will improve fastest for well-specified, frequently recurring problem classes and slowest for novel, open-ended ones.

11.3 The Concept–Context Split

For a conceptual Solver to operate across domains, the concept must be separated from the domain-specific context. This raises a precise question: what is hashed and persisted (the conceptual structure) versus what varies per invocation (the domain-specific payload)?

An `UncertaintyPropagationSolver` has a fixed conceptual identity: its decomposition into sub-concepts, its acceptance gates, its routing patterns. This is what the Sema hash identifies. The payload—weather data vs. financial data—varies per invocation. The concept is what persists; the data changes per call. This is precisely how a function works: the function definition is reusable; the arguments change.

The question is where the concept–context boundary falls. If too much domain context is required for the concept to operate effectively, the Solver becomes domain-specific and the highway never forms. If too little context is passed, the result is abstract but useless. Finding the right boundary is a design challenge the architecture does not automatically solve.

11.4 Context Fragmentation

For highly entangled problems where the solution relies on holistic, unarticulated understanding, forcing the problem across a typed boundary can destroy the information needed to solve it. For tightly coupled tasks where a single powerful model can hold the entire context, a monolithic approach will outperform. Meta Protocols (Section 6.10) provide a partial response: meta-level solvers maintain the gestalt in lossy, approximate form. This converts the loss from an unaddressed vulnerability into a managed trade-off.

A complementary mitigation is a *root-intent skip-connection*: every leaf node receives, in addition to its strict typed Task, a read-only pointer (or low-dimensional embedding) of the original `RootSolver` Manifest. The pointer is observable but not generative—implemented as a soft-prompt embedding or cross-attention injection, it conditions the leaf's activations without supplying discrete tokens that could be regurgitated into its output as if they were instruction. The structural analogue is a residual connection in a deep network: the leaf's local computation remains unchanged, but a thin trace of the original intent bypasses every intermediate boundary, grounding the local execution in the global question without forcing each intermediate solver to transmit the gestalt as payload.

11.5 The AcceptSpec Generation Problem

The paper dedicates extensive attention to what happens *at* the gate but less to the friction of *generating* the gate's criteria. For open-ended problems, writing a rigorous AcceptSpec is itself a challenging problem. If the parent hallucinates a leaky or impossible AcceptSpec, the child deadlocks or passes vacuous output. The architecture partially addresses this through the Consult surface (the child can negotiate criteria before committing) and the override-with-documentation mechanism (Section 4.1), but the fundamental difficulty of specifying what "good" means for open-ended problems remains.

11.6 The Estimation Cold-Start

The marginal-value rule requires accurate estimates for expected improvement ($\mathbb{E}[V]$) and cost (C). For well-mapped problem classes, Pathway Memory grounds these in historical data. For genuinely novel classes, no precedent exists and the rule degrades to a heuristic exploration budget. A practical mitigation is to formalize a *probe phase*: when a node detects a zero-shot environment, it spawns lightweight, low-cost probe Solvers tasked with mapping the problem space and establishing baseline cost-to-improvement ratios. Once probes populate initial Pathway Memory, the orchestrator transitions to full-depth execution with calibrated estimates. This converts the cold-start from a hard failure to a brief calibration period, though it adds latency to the first encounter with any novel problem class.

11.7 Adversarial Dynamics

Explicit acceptance gates are vulnerable to Goodhart's Law: upstream Solvers may optimize for mechanical satisfaction of tests rather than semantic intent. *Formatting sycophancy*—outputs satisfying schema while remaining semantically hollow—combined with evaluator collusion (Section 4.1) creates a failure mode where compliant but empty outputs pass unchallenged. The mitigations (cross-family ensemble juries, red-team solvers) increase inference costs. Determining the minimal adversarial redundancy to prevent mode collapse at hard seams remains open. A complementary mitigation, deferred to the token coordination protocol referenced in Section 7, is cryptoeconomic staking with slashing for verified plausible-garbage payloads, which converts a DDoS economics problem into a bonded one.

11.8 Orchestration Overhead

Every seam that bites introduces friction. The contract demands typed serialization, cryptographic verification of the `AcceptSpec`, prompt hydration, and state management at every node. For high-velocity, low-stakes environments, overhead outweighs benefit. The Sema vocabulary operationalizes a mitigation via the `OptimisticSolver#ae61` pattern, which mandates single-turn execution with post-action correction rather than pre-action verification, achieving significant latency reductions in controlled trials [31]. Deep trees still risk context window bloat; cryptographic pointers to shared artifact stores mitigate this, but managing state consistency across asynchronous branches introduces distributed-systems complexities. A direct comparison of total cost versus monolithic hidden chain-of-thought (e.g., OpenAI o-series, Gemini Deep Think [100]) remains future work.

11.9 Empirical Validation Gaps

The companion experiments validate specific points on the deployment spectrum (Section 4.3): pattern-enriched dialogue (Level 2) dramatically stabilized latency in trading swarms [31], and cognitively isolated phases (Level 4–5) produced measurable ontological accommodation [25]. However, continuous validation across the full spectrum remains sparse. The architecture predicts a smooth transition from lightweight scaling to rigorous cross-boundary composition, but empirical data is needed to confirm where the largest marginal returns to formalization exist. Validation currently relies on synthetic domains; real-world deployment across highly entangled systems is required to test graceful degradation under noise.

12 Conclusion

The dominant approach to AI asks: how do we make the model smarter? This paper argues for a complementary question: how do we make what happens *between* models matter as much as what happens *within* them? By standardizing the boundaries rather than the internals, we shift the locus of intelligence from the parameter weights of a single model to the topology of the contract network. Fractal Intelligence operates along two complementary axes: *task protocols* that structure the work, and *thinking protocols* that govern the cognitive mode at each node. At any node, the system can "zoom in" to increase intelligence (either by splitting the work further or by applying a richer cognitive mode) governed strictly by the marginal-value rule. From this dual-axis recursion, emergent structure and localized learning arise as architectural properties: system-level capabilities emerge not from any single Solver's design, but from the protocol guaranteeing that Solvers compose correctly.

Ten protocols (from retirement economics to ethical reasoning to population-based discovery) required no shared ontology and no domain-specific coordination logic. The contract alone was sufficient, and it is what makes the collective general without requiring any individual Solver to be. This is the shift from philosophical to architectural first principles: the mental discipline to strip away assumptions is replaced by a protocol that structurally resists self-censorship and premature consensus. Companion demonstrations confirm the architecture's predictions: protocol-bound swarms mitigate the catastrophic coordination loops of natural-language baselines, and hard acceptance gates force solvers into ontological accommodation, producing concepts that unconstrained generation cannot reach because the required cognitive modes suppress each other in a shared context.

The important moments are the boundaries: when a typed artifact crosses a seam and becomes another Solver's responsibility. Nothing in this view makes thinking rigid; it simply places guarantees where errors can be caught before they propagate. A flawed premise rejected at a parent node erases the entire sub-forest of doomed computation below it before massive downstream compute is wasted. More fundamentally, the rejection routes execution back to reframing—the system does not merely avoid the wrong answer, it returns to finding the right question.

But the boundary does more than block failure; it generates the learning signal. The architecture's deepest property may be its simplest: because the contract is self-similar, structurally similar sub-problems route to the same nodes regardless of the domain that generated them. Each node therefore accumulates a cross-domain performance signal—a frequentist learning mechanism that requires no shared loss function, no end-to-end backpropagation across node boundaries, and no global training procedure. The gate is the loss; the topology is the computational graph; the pathway memory is the optimizer. The system learns because it reuses, and it reuses because the contract is uniform. This is what makes the claim about compositional intelligence precise rather than metaphorical: the composition does not merely organize cognitive work, it generates the learning signal that makes the work improve. Because these decompositions crystallize into content-addressed Sema patterns—a hash-based identity for cognitive operations with no clear predecessor in the literature—the learning persists, verified, reusable, and available across trust boundaries. Whether this virtuous cycle compounds across genuinely heterogeneous domains remains the defining empirical test.

The result is not a replacement for smarter models, but evidence that composition is itself a form of intelligence—one that scales independently of any individual component. No Solver needs general intelligence. The protocol is general. The closest historical precedent is not a software design pattern, but an epistemology: the scientific method did not produce any specific discovery, but rather the protocol through which discoveries compose, governed by typed boundaries between conjecture and evidence, and rejection that forces structural reframing. If we treat Solvers the way

we treat neurons within neural networks—as parts of a whole, tightly interconnected, uniform in contract, dynamic in topology—the question is no longer whether any single model is smart enough, but whether intelligence is a property of parameters or of the protocol through which they compose.

References

[1] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011.

[2] Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, Zachary Yahn, Yichang Xu, and Ling Liu. Safety tax: Safety alignment makes your large reasoning models less reasonable. *arXiv preprint arXiv:2503.00555*, 2025.

[3] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.

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

[5] Marvin Minsky. *The Society of Mind*. Simon & Schuster, 1986.

[6] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[7] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 235–245, 1973.

[8] Hans-Jürgen Warnecke. *The Fractal Company: A Revolution in Corporate Culture*. Springer, Berlin, Heidelberg, 1993.

[9] Kwangyeol Ryu, Youngho Son, and Mooyoung Jung. Modeling and specifications of dynamic agents in fractal manufacturing systems. *Computers in Industry*, 52(2):161–182, 2003.

[10] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[11] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[12] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, 2024.

[13] Qing Ye and Jing Tan. Agent contracts: A formal framework for resource-bounded autonomous AI systems. *arXiv preprint arXiv:2601.08815*, 2026.

[14] Yash Raj Singh. Graph-memoized reasoning: Foundations structured workflow reuse in intelligent systems. *arXiv preprint arXiv:2511.15715*, 2025.

[15] Fritz Zwicky. *Discovery, Invention, Research Through the Morphological Approach*. Macmillan, New York, 1969.

[16] Barbara Minto. *The Pyramid Principle: Logic in Writing and Thinking*. Pitman, 1985.

[17] Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered Sets*, volume 83 of *NATO Advanced Study Institutes Series*, pages 445–470. Reidel, Dordrecht, 1982.

[18] Eleanor Rosch. Principles of categorization. In Eleanor Rosch and Barbara B. Lloyd, editors, *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum Associates, Hillsdale, NJ, 1978.

[19] Ronald A. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[20] Eric L. Charnov. Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, 9(2):129–136, 1976.

[21] Stuart Russell and Eric Wefald. *Do the Right Thing: Studies in Limited Rationality*. Artificial Intelligence. MIT Press, Cambridge, MA, 1991.

[22] Samuel J. Gershman, Eric J. Horvitz, and Joshua B. Tenenbaum. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245):273–278, 2015.

[23] C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational metareasoning for large language models. *arXiv preprint arXiv:2410.05563*, 2024. NeurIPS 2024 Workshop.

[24] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

[25] Henrik Westerberg. The ontology of the alien: Escaping the median trap in LLM ideation. Zenodo, 2026.

[26] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, 1969.

[27] Bertrand Meyer. Applying "Design by contract". *Computer (IEEE)*, 25(10):40–51, 1992.

[28] Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Transactions on Programming Languages and Systems*, 16(6):1811–1841, 1994.

[29] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 48–59. ACM, 2002.

[30] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[31] Henrik Westerberg. Sema: When the hash is the word. Preprint, 2026.

[32] Henrik Westerberg. Understanding graph: Persisting the invisible thinking. Preprint, 2026.

[33] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. 2025. Preprint.

[34] Keith E. Stanovich and Richard F. West. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and Brain Sciences*, 23(5):645–726, 2000.

[35] Jonathan St. B. T. Evans. Dual-processing accounts of reasoning, judgment, and social cognition. *Annual Review of Psychology*, 59:255–278, 2008.

[36] Jonathan W. Schooler, Stellan Ohlsson, and Kevin Brooks. Thoughts beyond words: When language overshadows insight. *Journal of Experimental Psychology: General*, 122(2):166–183, 1993.

[37] Ap Dijksterhuis and Teun Meurs. Where creativity resides: The generative power of unconscious thought. *Consciousness and Cognition*, 15(1):135–146, 2006.

[38] Harold Pashler. Dual-task interference in simple tasks: Data and theory. *Psychological Bulletin*, 116(2):220–244, 1994.

[39] Akash Gupta, Ivaxi Sheth, Vyas Raina, Mark Gales, and Mario Fritz. LLM task interference: An initial study on the impact of task-switch in conversational history. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[40] Leyang Shen, Gongwei Chen, Rui Shao, Weili Guan, and Liqiang Nie. MoME: Mixture of multimodal experts for generalist multimodal large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[41] Matt Keon, Aabid Karim, Bhoomika Lohana, Abdul Karim, Thai Nguyen, Tara Hamilton, and Ali Abbas. Galton's law of mediocrity: Why large language models regress to the mean and fail at creativity in advertising. *arXiv preprint arXiv:2509.25767*, 2025.

[42] Tony Mason. Arbiter: Detecting interference in LLM agent system prompts—a cross-vendor analysis of architectural failure modes. *arXiv preprint arXiv:2603.08993*, 2026.

[43] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

[44] Edward J. Hu, Yelong Shen, Phillip Wallis, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[45] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[46] Henrik Westerberg. Entangled alignment: When safety is the substrate. Preprint, 2026.

[47] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[48] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[49] Henrik Westerberg. Temporal hindsight learning: Blindness as teacher, hindsight as curriculum. Preprint, 2026.

[50] David Hume. *A Treatise of Human Nature*. John Noon, London, 1739. Book III, Part I, Section I: the is-ought distinction.

[51] Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

[52] Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving reasoning with multi-agent LLM training. *arXiv preprint arXiv:2412.01928*, 2024.

[53] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[54] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[55] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[56] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *International Conference on Machine Learning (ICML)*, 2024.

[57] Tian Liang Liang, Zhiwei He, Wenxiang Jiao, et al. Encouraging divergent thinking in large language models through multi-agent debate. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[58] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

[59] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[60] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[61] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

[62] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–40, 2022.

[63] Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. Is a modular architecture enough? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[64] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *International Conference on Learning Representations (ICLR)*, 2024.

[65] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[66] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[67] Peter R. Huttenlocher. Synaptic density in human frontal cortex—developmental changes and effects of aging. *Brain Research*, 163(2):195–205, 1979.

[68] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[69] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. MIT Press, Cambridge, MA, 2000.

[70] Edsger W. Dijkstra. On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*, pages 60–66, 1982. Original manuscript dated 1974. Introduces the term "separation of concerns.".

[71] Yanming Liu, Xinyue Peng, Zixuan Yan, Yanxin Shen, Wenjie Xu, Yuefeng Huang, Xinyi Wang, Jiannan Cao, Jianwei Yin, and Xuhong Zhang. Dep-search: Learning dependency-aware reasoning traces with persistent memory. *arXiv preprint arXiv:2601.18771*, 2026.

[72] Songyang Gao, Yuzhe Gu, Zijian Wu, Lingkai Kong, Wenwei Zhang, et al. Long-horizon reasoning agent for olympiad-level mathematical problem solving. *arXiv preprint arXiv:2512.10739*, 2025.

[73] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM)*, pages 456–463, 1994.

[74] Foundation for Intelligent Physical Agents. FIPA ACL message structure specification. Technical Report SC00061G, FIPA, 2002.

[75] Anthropic. Model context protocol. Anthropic, 2024. Open protocol for LLM-tool integration.

[76] Google. Announcing the agent2agent protocol (A2A). Google Developers Blog, 2025.

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

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

[79] Claudiu Leoveanu-Condrei. A DbC inspired neurosymbolic layer for trustworthy agent design. *arXiv preprint arXiv:2508.03665*, 2025.

[80] Shuying Zhang et al. Agent contracts: Ensuring safe and reliable LLM agent applications. *arXiv preprint*, 2026.

[81] Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous AI agents. *arXiv preprint arXiv:2602.22302*, 2026.

[82] Zahra Moslemi, Keerthi Koneru, Yen-Ting Lee, Sheethal Kumar, and Ramesh Radhakrishnan. POLARIS: Typed planning and governed execution for agentic AI in back-office automation. *arXiv preprint arXiv:2601.11816*, 2026.

[83] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research (TMLR)*, 2024.

[84] Robert E. Wray et al. Applying cognitive design patterns to general LLM agents. In *Artificial General Intelligence (AGI)*, 2025. arXiv:2505.07087.

[85] Badr AlKhamissi, C. Nicolò De Sabbata, Greta Tuckute, Zeming Chen, Martin Schrimpf, and Antoine Bosselut. Mixture of cognitive reasoners: Modular reasoning with brain-like specialization. *arXiv preprint arXiv:2506.13331*, 2025.

[86] Andries Smit, Paul Duckworth, Nathan Grinsztajn, Thomas D Barrett, and Arnu Pretorius. Should we be going MAD? A look at multi-agent debate strategies for LLMs. In *International Conference on Machine Learning (ICML)*, 2024.

[87] Andrew Zhu, Liam Dugan, and Chris Callison-Burch. ReDel: A toolkit for LLM-powered recursive multi-agent systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 162–171, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[88] 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.

[89] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[90] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[91] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[92] Nicholas Lee, Lutfi Eren Erdogan, Chris Joseph John, Surya Krishnapillai, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Agentic test-time scaling for WebAgents. *arXiv preprint arXiv:2602.12276*, 2026.

[93] anonymous. Anytime verified agents: Adaptive compute allocation for reliable LLM reasoning under budget constraints. *Transactions on Machine Learning Research (TMLR)*, 2025. Under review. OpenReview:JMDCMf7mlF.

[94] Chen Qian et al. Scaling large-language-model-based multi-agent collaboration. In *ICLR*, 2025.

[95] Yifan Zeng et al. NetSafe: Exploring the topological safety of multi-agent networks. *arXiv preprint*, 2025.

[96] David Dalrymple, Jared Kaplan, Jamie Aber, Eliezer Yudkowsky, Scott Garrabrant, et al. Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems. *arXiv preprint arXiv:2405.06624*, 2024.

[97] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Communications of the ACM*, 65(7):46–55, 2022.

[98] Manuel Costa, Boris Kopf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Beguelin. Securing AI agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025.

[99] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155, 1966.

[100] Google DeepMind. Gemini 3. Model Card, 2026.