Metacognitive Architecture & Instruction Layers: A Modular Governance Framework for Reasoning Systems

Francisco Revelles

December 9, 2025

Abstract

Large language models demonstrate increasingly sophisticated reasoning but lack persistent, hierarchical governance over cognition, continuity, and anti-fabrication safeguards. This paper introduces the *Metacognitive Architecture & Instruction Layers*, a modular framework designed to implement an Instructional Operating System (IOS) for AI systems. By decoupling content-production layers (L0–L1) from control-governance layers (L2–L3), the architecture enables structured, auditable, and domain-isolated reasoning suitable for high-stakes, long-horizon applications.

1 Introduction

The recent advances in large language models (LLMs) have produced systems capable of generating elaborate chains of reasoning on demand. However, these systems still lack an explicit, persistent structure for governing how reasoning is conducted over time, across domains, and under safety or reliability constraints. Prompts can request a specific style or discipline of reasoning, but the model does not maintain a non-overridable hierarchy of rules that outlives any single exchange.

This paper proposes the *Metacognitive Architecture & Instruction Layers*, a four-layer governance framework designed to act as an Instructional Operating System for LLM-based systems. The architecture decomposes cognition into:

- L0: Global Cognitive Rules,
- L1: Domain Reasoning Layers,
- L2: Project Governance, and
- L3: Testing & Enforcement.

The core idea is to separate *content* (what the model says) from *control* (how and under which constraints it is allowed to say it). This decoupling enables structured continuity, explicit anti-fabrication checks, and isolation between domains such as legal reasoning, clinical support, and systems design.

Contributions. The main contributions of this work are:

(1) a conceptual and formal definition of a four-layer metacognitive architecture for LLM-governed reasoning systems;

(2) a control/content decoupling scheme in which L0–L1 define reasoning content and L2–L3 define governance and enforcement;

(3) worked examples illustrating how the architecture can be instantiated in practical domains requiring continuity, procedural rigor, and cross-domain isolation.

The remainder of this paper is organized as follows. Section **??** situates the work within related efforts in tool-augmented LLMs, guardrails, and system prompts. Section **??** introduces the four layers and their interactions. Section **??** presents a formalization of the architecture. Section **??** gives worked examples, and Section **??** discusses connections to prior work. We conclude in Section **??**. As a working reference, we denote this framework as *Metacognitive Architecture & Instruction Layers* [**?**].

2 Background and Motivation

This section situates the proposed architecture relative to existing approaches for controlling and structuring LLM behavior. We highlight three threads: (i) prompt engineering and system prompts, (ii) tool and agent frameworks, and (iii) safety, guardrail, and verification work.

Prompt engineering and system prompts. Much of the practical control over LLM behavior today is expressed through long system prompts, templates, and few-shot examples. These techniques can encode style, constraints, or domain-specific rules, but they lack persistent structure: there is no native separation between global rules and per-project instructions, and no explicit notion of testing and enforcement layers.

Tools and multi-agent frameworks. Agent and tool frameworks introduce explicit roles, enabling models to call external tools (search, code execution, retrieval) and to subdivide tasks across multiple agents. While these frameworks introduce a form of structure, they typically do not formalize a layered governance model over cognition itself. The question of *who* enforces global rules and *how* cross-domain isolation is guaranteed remains loosely specified.

Safety, guardrails, and verification. Guardrail systems and safety policies attempt to prevent models from emitting harmful or disallowed content. Verification-oriented work checks individual answers or proofs for correctness. These directions address important aspects of risk, but usually at the level of content filtering or post-hoc checking, rather than a general metacognitive architecture that governs how reasoning is structured from the outset.

The Metacognitive Architecture & Instruction Layers proposed in this paper can be seen as a unifying abstraction over these threads. It organizes the control space into a fixed hierarchy of layers (L0–L3) that can host prompts, tools, and guardrails in a principled manner.

3 The Layered Architecture

The architecture decomposes cognition and governance into four layers, L0 through L3, forming a fixed hierarchy. Each layer has a distinct responsibility and a defined interface to the others.

L0: Global Cognitive Rules. L0 encodes non-overridable constraints on reasoning. Examples include: (1) an abstract $\rightarrow$ domain $\rightarrow$ task reasoning flow; (2) explicit separation of fact, inference, and unknown; and (3) anti-fabrication discipline. Intuitively, L0 states *how* all reasoning must be structured, regardless of domain.

L1: Domain Reasoning Layers. Each domain (e.g., legal, medical, AI-systems design) receives its own L1 layer. An L1 layer specializes the global rules to domain-specific concepts, ontologies, and validity criteria. Domain isolation is enforced by ensuring that L1 layers cannot directly modify L0 or other L1 layers.

L2: Project Governance. L2 operates as an orchestrator and lifecycle manager. It is responsible for:

- installing and updating L1 layers for specific projects;
- enforcing project isolation (separating one active project from another); and
- preserving continuity, such as ensuring that earlier commitments or constraints are remembered and respected.

L3: Testing & Enforcement. L3 represents the veto and validation layer. It receives proposed outputs together with their reasoning traces and evaluates them for compliance with L0–L2. L3 can:

- reject outputs that violate anti-fabrication rules;
- flag incomplete reasoning or missing conditions;
- enforce readiness checks before an answer is released.

Layer Stack Diagram

Figure **??** depicts the layer stack. The lower layers (L0–L1) are content-producing, while the upper layers (L2–L3) are control-governing.

Later sections will introduce a more formal model of these layers, as well as concrete instantiations in applied settings.

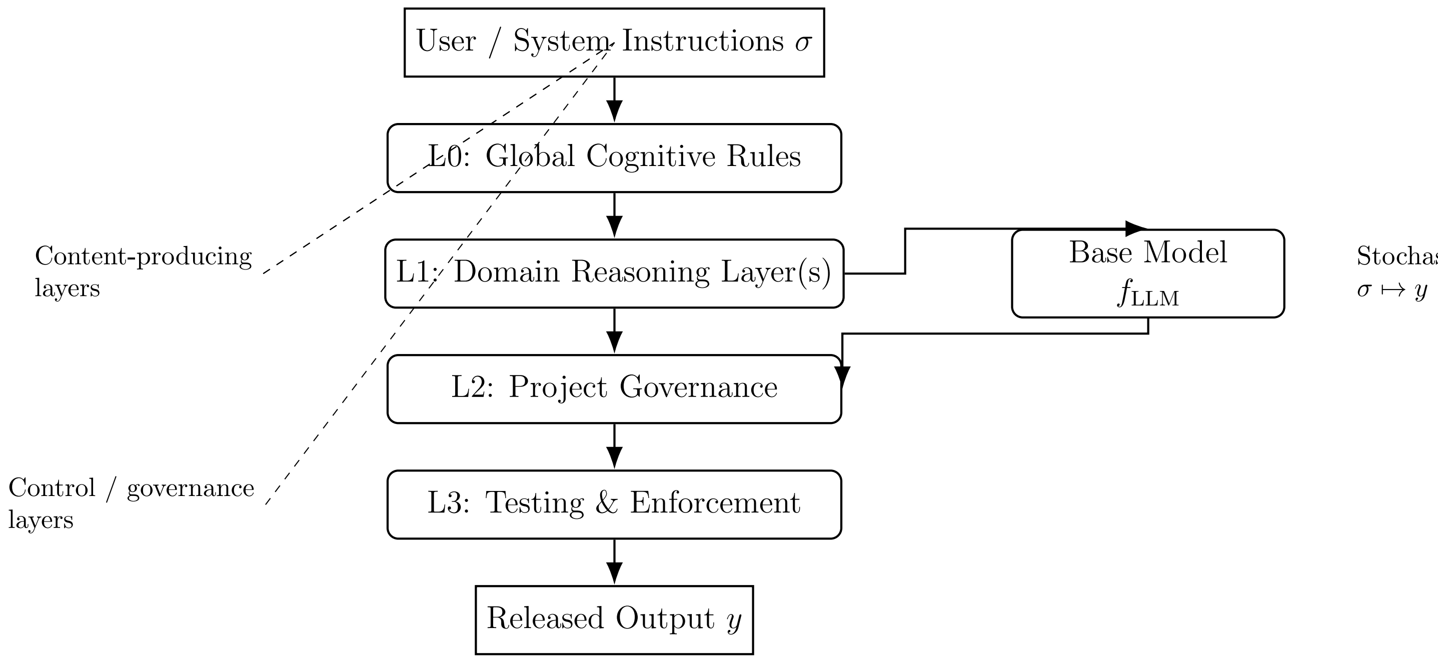
User / System Instructions σ
L0: Global Cognitive Rules
L1: Domain Reasoning Layer(s)
L2: Project Governance
L3: Testing & Enforcement
Released Output y
Base Model f_{LLM}
Stochas
$\sigma \mapsto y$
Content-producing layers
Control / governance layers

Figure 1: Metacognitive Architecture as a layered pipeline. L0–L1 transform and constrain instructions before they reach the base model f_{LLM}. L2 manages project-level governance and continuity, while L3 performs testing and enforcement before any output y is released.

4 Formal Model

This section presents a more structured formalization of the metacognitive architecture. The goal is not to fix a single canonical formalism, but to show that L0–L3 can be modeled as operators over instructions, states, and outputs, with clearly separated roles for content and control.

Basic objects

Let:

- Σ be the set of possible instructions (prompts, policies, context strings),
- $\mathcal{O}$ be the set of possible outputs, and
- $\mathcal{S}$ be the space of internal states of the system (including project history, active domains, and memory).

A bare language model can be idealized as a (possibly stochastic) map

$$f_{\text{LLM}} : \Sigma \times \mathcal{S} \to \mathcal{O} \times \mathcal{S}.$$

We make no assumption here about how f_{LLM} is implemented; it could be any autoregressive or retrieval-augmented architecture.

Layer operators

The metacognitive architecture introduces four classes of operators, L_0, L_1, L_2, L_3, each acting on (σ, s) or on candidate outputs.

Definition 1 (Global cognitive layer L_0)**.** *The L0 layer is an operator*

$$L_0 : \Sigma \times \mathcal{S} \to \Sigma \times \mathcal{S}$$

that enforces global reasoning constraints. Typical effects include: normalizing the instruction into an abstract → domain → task structure, appending global anti-fabrication rules, or imposing a fact / inference / unknown separation discipline.

Definition 2 (Domain reasoning layers L_1^d)**.** *For each domain d in some index set $\mathcal{D}$ (e.g., legal, medical, AI-systems), there is a domain-specific layer*

$$L_1^d : \Sigma \times \mathcal{S} \to \Sigma \times \mathcal{S}$$

that specializes instructions to domain d, injecting ontologies, local constraints, and validity conditions. A configuration may enable a subset $\mathcal{D}_{active} \subseteq \mathcal{D}$.

Definition 3 (Project governance layer L_2)**.** *The L2 layer manages project-level state. Given a project identifier p from some set of projects $\mathcal{P}$, we write*

$$L_2^p : \Sigma \times \mathcal{S} \times \mathcal{D}_{active} \to \Sigma \times \mathcal{S}.$$

Intuitively, L_2^p:

(a) selects which domain layers L_1^d are active for project p,

(b) re-applies project constraints (e.g., prior commitments, safety requirements),

(c) maintains continuity by updating the project state component of $\mathcal{S}$.

Definition 4 (Testing and enforcement layer L_3)**.** *Given a candidate output $y \in \mathcal{O}$, an associated reasoning trace τ, and state $s \in \mathcal{S}$, the L3 layer produces an enforcement decision*

$$L_3 : \mathcal{O} \times \mathcal{T} \times \mathcal{S} \to \{accept, revise, reject\}$$

for some space of traces $\mathcal{T}$. In the case revise, L_3 may also emit a refined instruction σ_{rev} to be fed back into the pipeline.

Execution pipeline

A single reasoning episode under project $p \in \mathcal{P}$ proceeds as follows. Given an initial instruction $\sigma_0 \in \Sigma$ and state $s_0 \in \mathcal{S}$:

(i) **Global normalization (L0).**

$$(\sigma_1, s_1) = L_0(\sigma_0, s_0).$$

(ii) **Domain specialization (L1).** Let $\mathcal{D}_{\text{active}}$ be the set of domain layers chosen for project p. We apply them in some admissible order (e.g., a fixed total order or a partial order resolved by L_2):

$$(\sigma_2, s_2) = \Big(\prod_{d \in \mathcal{D}_{\text{active}}} L_1^d \Big)(\sigma_1, s_1).$$

(iii) **Project governance (L2).**

$$(\sigma_3, s_3) = L_2^p(\sigma_2, s_2, \mathcal{D}_{\text{active}}).$$

(iv) **Base model call.**

$$(y, s_4) = f_{\text{LLM}}(\sigma_3, s_3).$$

During this step, a reasoning trace τ is also accumulated (e.g., chain-of-thought, tool calls, or intermediate states).

(v) **Testing and enforcement (L3).** L3 evaluates (y, τ, s_4):

$$d = L_3(y, \tau, s_4).$$

If $d = \text{accept}$, the system releases y. If $d = \text{reject}$, the output is withheld. If $d = \text{revise}$, the system obtains a revised instruction σ_{rev} and may re-enter the pipeline at step (i) or (ii).

We call this composition the *governed reasoning pipeline* for project p.

Separation of content and control

The architecture prescribes a structural separation:

- Content-producing components are f_{LLM} together with the instruction transformations in L0 and L1.
- Control-governing components are L_2 and L_3, which decide which domains are active, how project state evolves, and whether candidate outputs are acceptable.

This separation can be made precise by viewing L_2 and L_3 as operating on a *meta-level* configuration that is not writable by L_1^d or f_{LLM} directly. One can then reason about invariants such as:

- domain isolation (no project may activate conflicting domain rules simultaneously);
- anti-fabrication discipline (L3 rejects outputs that violate L0's fact / inference / unknown separation);
- continuity (L2 re-applies project constraints across episodes).

These invariants are enforced by construction at the level of the pipeline, rather than as ad-hoc prompt instructions. A more detailed formal treatment could use transition systems or modal logics to reason about reachable states and safety properties, but the operator view above is sufficient to capture the intended governance semantics.

5 Worked Examples and Applications

To illustrate the architecture, we outline several scenarios in which L0–L3 provide governance that is difficult to obtain from prompts alone. The examples are intentionally heterogeneous to emphasize cross-domain isolation and continuity.

Example 1: Legal reasoning vs. medical reasoning. In one project, an L1 layer is instantiated for legal reasoning under a specific jurisdiction. In a separate project, another L1 layer is used for clinical communication. L2 ensures that domain-specific rules (e.g., evidentiary standards) are not silently imported into clinical explanations, and vice versa. L3 enforces that any legal answer carries an explicit separation between fact, inference, and unknown.

Example 2: Long-horizon project continuity. Consider a multi-week technical project where an AI assistant helps refine a system design. L2 maintains project-level constraints (such as a chosen architecture style or safety requirement) and re-applies them across sessions. L3 rejects outputs that regress on previously agreed constraints, forcing the system to either justify deviations or stay consistent.

Example 3: Anti-fabrication discipline. In a research or compliance setting, L0 imposes a strict fact/inference/unknown separation. When the base LLM proposes an answer, L3 inspects the trace for unsupported claims. If unresolved gaps remain, L3 can either request additional evidence (via tools) or downgrade the answer to an explicitly uncertain status, rather than silently hallucinating details.

These examples are not exhaustive, but they demonstrate how a layered metacognitive architecture can turn informal prompting patterns into an explicit, auditable control structure.

6 Related Work

The proposed architecture intersects multiple research threads, including tool-augmented LLMs, AI safety and alignment, program synthesis and verification, and cognitive architectures.

Tool-augmented and agentic LLMs. Work on tool use and agents gives LLMs the ability to call external functions, maintain working memory, or coordinate multiple roles. However, these frameworks often treat high-level governance as an implementation detail rather than a first-class object. The present architecture can be viewed as a way to factor such systems into explicit, named layers with clearly articulated responsibilities.

Safety and guardrails. Guardrail systems define policies about allowed content and sometimes about required checks. Our L3 layer plays a complementary role: it provides a structural place where policy enforcement, readiness checks, and verification can be composed and reasoned about.

Cognitive architectures. Classical cognitive architectures and metacognition research study how systems can monitor and regulate their own reasoning processes. The Metacognitive Architecture & Instruction Layers reinterprets these ideas in the specific context of LLM-governed systems, emphasizing the need for persistent, hierarchical instruction governance.

A more detailed comparison with specific systems and frameworks can be developed as the architecture is instantiated in concrete implementations.

7 Conclusion

This paper introduced the Metacognitive Architecture & Instruction Layers, a four-layer governance framework designed to act as an Instructional Operating System for LLM-based reasoning systems. By separating content-producing layers (L0–L1) from control-governing layers (L2–L3), the architecture creates space for continuity, cross-domain isolation, and explicit anti-fabrication enforcement.

Several directions remain for future work. On the theoretical side, richer formal models could clarify the contracts between layers and enable partial verification. On the practical side, implementing the architecture in real-world systems—and stress-testing it on high-stakes tasks—will be essential for understanding its strengths and limitations.

Ultimately, the goal is not to freeze a single configuration of rules, but to provide a reusable, extensible scaffolding for metacognitive control. As LLMs become more capable and are embedded in safety-critical workflows, explicit, layered governance over their reasoning will become not just desirable, but necessary.