Neurosymbolic Artificial Intelligence
XX(X):2–34

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Operability as Structural Grounding: Conditions for Epistemic Appropriation in LLMs

Andrea Gasparro[1] and Paolo Ciancarini[2]

Abstract

Large Language Models necessarily hallucinate: Kalai and Vempala (2024) prove that any well-calibrated model must assign non-zero probability to unseen fact combinations, making confabulation a mathematical consequence of statistical inference. Whether LLMs' internal representations constitute genuine world models Li et al. (2023); Gurnee and Tegmark (2024) or not Kang et al. (2024), a gap remains: the model has no mechanism to distinguish, in its outputs, grounded facts from statistical interpolation. Since LLMs cannot reliably introspect their own reasoning Turpin et al. (2023), we hypothesize that this epistemic gap manifests differently depending on the structural properties of the domain in which a generative system operates. We introduce a mechanistic taxonomy distinguishing Structural, Epistemic, and Training-induced hallucinations by their architectural origin. We then build a neuro-symbolic framework, to prove that if LLMs intrinsically produce Epistemic Hallucinations, in domains with three structural features (executability, semantic persistence, and interpretable feedback) that allow neural models to generate forward and appropriate their outputs backward Dherin et al. (2025),they encounter the behavioral consequences of their generation as a means of epistemic grounding. We evaluate this framework across both non-thinking and thinking-augmented architectures of different models (including Claude Sonnet 3.5/4.6 and Gemini 2.5/3 Flash) in two distinct neuro-symbolic configurations: maze-solving and computational delegation via an Answer Set Programming solver (Clingo). Across all models and domains, formal syntax alone fails to ground reasoning: without execution, code-based reasoning underperforms natural language, and in thinking-augmented models, it leads to unfaithful reasoning traces or substrate exhaustion. However, when closed through the external symbolic verifier, performance consistently reaches deterministic fidelity (e.g., jumping from 56% to 100% accuracy in maze-solving, and 13% to 97% in ASP), suggesting that the advantage lies not in formal syntax, but in the closed symbolic verification loop in operable domains—as a mean of epistemic appropriation.

Keywords
Neuro-Symbolic AI, Explainable AI, Epistemic Hallucinations, Epistemic Grounding, Knowledge Representation, Operability

1 Introduction

1.1 The Paradox of Necessary Hallucination

Large Language Models produce text that is statistically plausible but not necessarily true—a phenomenon broadly termed *hallucination*. The dominant research narrative treats hallucination as a defect to be mitigated: through retrieval-augmented generation, improved training procedures, or better prompting strategies. However, a growing body of theoretical work challenges this framing at its foundation.

Kalai and Vempala (2024) prove that any language model maintaining a well-calibrated probability distribution must hallucinate with non-negligible probability on queries about rare or unseen fact combinations. The argument rests on a fundamental tradeoff: calibration requires the model to assign probability mass to novel combinations—proportionally to the fraction of information appearing only once in training—which necessarily entails generating plausible statements about facts the model has never encountered. Kalai et al. (2025) further demonstrate that standard loss functions exacerbate this: cross-entropy penalizes explicit uncertainty relative to confident guessing, creating what they term the "risk of hedging"—the model is trained to guess confidently rather than express doubt.

Research on hallucination has evolved through recognizable phases: from context-grounding approaches that assumed external information would anchor outputs to factual sources Lewis et al. (2020), through calibration studies revealing that models often "know what they know" internally yet fail to communicate this reliably Kadavath et al. (2022), to the current theoretical understanding that hallucination is architecturally intrinsic. If hallucination cannot be eliminated, the question becomes: *under what conditions can it be harnessed as a feature of the inference process?*

1.2 The Gap

Most existing work focuses on *detecting* or *reducing* hallucinations through prompting techniques, retrieval augmentation, or improved training data. Existing taxonomies classify hallucinations by their observable output—factuality versus faithfulness Huang et al. (2025), intrinsic versus extrinsic Cossio (2025)—or by the pipeline stage at which they arise. While useful for detection, these classifications do not identify the

[1]Università di Bologna, Italy

Corresponding author:
Andrea Gasparro, Università di Bologna, Bologna, Italy.
Email: andrea.gasparro@gmail.com

architectural features that distinguish Epistemic hallucinations, nor do they trace such hallucinations to their root cause: the model's own knowledge representation.

We propose a mechanistic taxonomy that distinguishes hallucinations by their architectural origin—Structural, Epistemic, and Training-induced—with the specific aim of highlighting how Epistemic hallucinations constitute a fundamentally different category. The architectural root is that the same parameters encode both verified facts and plausible interpolations, making the two indistinguishable at inference time. The model cannot track the provenance of its own knowledge—it does not "know" whether it knows or merely interpolates.

1.3 Thesis

Our starting point is an observation about where current models succeed and where they fail: they reason reliably when the artifacts they generate during a process retain their meaning across its steps, and they drift—into plausible but ungrounded interpolation—when that meaning is left to be reconstructed internally. The distinguishing factor is not whether the output carries formal structure but whether its *semantics*—the symbolic meaning of the artifact—is carried forward, encapsulated in formal structure. Two clarifications keep this from collapsing into weaker, already-known claims. First, formal syntax helps the model recognize patterns and produce well-formed artifacts, yet syntax alone is inert with respect to grounding. Second—less obviously—it is not iteration or self-revision as such that grounds, not even one that carries a complete trace of its own previous steps: DeVilling (2025) shows that recursive self-evaluation without external feedback yields reformulation rather than progress, because a model that treats its own prior outputs as evidence conserves its initial uncertainty—no new information enters the sequence. What grounds is semantics crystallized in the produced artifact's syntax: what re-enters is then not the model's narration of what it did, but the deterministic consequence of what it actually produced—so that each successive inference is strongly conditioned by the specific properties of the artifact, rather than a purely statistical inference. This is not a property a formalism possesses intrinsically: natural language does not afford it, and code—though it affords it strongly—does not suffice on its own, as the gap between our unexecuted and executed conditions shows. What matters is not the choice of formalism but whether the process is structured so that meaning is realized and carried during generation.

What we set out to study is therefore not a new method for making models more accurate, nor a new tool to attach to inference, but the inference process itself—and specifically the structural characteristics a domain, or a process constructed over it, must exhibit for generation to stay grounded across its steps. This reframes the question the field usually asks. Work on execution feedback and tool use asks how to *improve* grounding within a fixed setup; we ask the prior question of which structural conditions of the model–mechanism–domain configuration make grounding occur *at all*, and we treat the answer as a property to be characterized rather than a technique to be optimized.

We argue that for Epistemic Hallucinations—those arising from the model's inability to track knowledge provenance—the solution is not better training or more data,

but *Operability*: a structural property of the model–mechanism–domain configuration, present when executability, semantic persistence, and interpretable feedback are jointly realized in a closed loop (Def. 1). In its paradigmatic form this loop is execution: when the model's output is run by a deterministic oracle, its *behavior* becomes the ground truth that the model cannot produce internally. Critically, this is not an external correction bolted onto an otherwise unchanged model. Dherin et al. (2025) establish that context entering a transformer is functionally equivalent to a transient, low-rank modification of the model's own weights for the next generation: feedback that has re-entered the context does not sit outside the model as a verdict on its output, but reparametrizes the computation that produces the following artifact. The correction is internalized, not imposed. This result, however, settles only *that* re-entry is consequential, not *which* re-entry grounds: the same mechanism operates identically for retrieved documents and for the model's own narration of its reasoning — the latter being the case, noted above, in which no new information enters and the model's uncertainty is merely conserved. What separates grounding from mere augmentation is therefore the content that re-enters; and only when the loop satisfies the three conditions of Section 3 is that content the deterministic consequence of the model's own externalized output. Execution then does more than verify: it transforms the inferential surface from which the next output is generated. This is the mechanical substrate of epistemic appropriation*: the model does not merely receive a verdict on its output, it operates from a context in which the consequences of that output have become functionally constitutive of its subsequent inference. This transforms the epistemological status of the output: what is merely generated remains epistemically opaque; what is executed, observed and integrated becomes epistemically owned.

Where Operability is present, hallucination becomes a form of productive exploration: the model generates candidate solutions that are iteratively tested and refined against concrete ground truth. Where it is absent — in domains where the external verifier does not preserve the output's semantics across iterations — hallucination renders outputs epistemically opaque, because no mechanism exists to distinguish truth from plausible confabulation.

The verification component in this framework functionally plays the same role as a Knowledge Representation system. In our experiment, the Python interpreter serves this role: it encodes the semantics of the programming language as domain constraints, performs deterministic symbolic computation, and returns interpretable feedback. But the interpreter is one instance of a broader pattern. An OWL reasoner checking the consistency of generated medical assertions, an ASP solver verifying that a proposed logistics plan satisfies domain constraints, a SPARQL engine validating generated knowledge graph triples against an existing ontology—each provides the same three

*We use 'appropriation' as a technical term defined operationally by the three structural conditions of Definition 1, not as a phenomenological category. The closest precedent is the pragmatist sense of taking-up-and-making-one's-own (cf. Brandom 1994, Haugeland 1998); we adopt the term for its semantic match to the structural property — output becoming functionally constitutive of subsequent inference — without committing to the broader philosophical machinery.

properties: executability, semantic persistence, and interpretable feedback. This positions KR not as an alternative to probabilistic generation, but as its necessary epistemic complement, allowing Operability as post-generative grounding. This complementarity carries a direct implication for explainability. Chain-of-thought verbalization, the most widely used surface for inspecting LLM reasoning, has been shown to be a plausible but unfaithful narrative of the underlying computation Turpin et al. (2023): the model's verbal account does not describe its inference, it reconstructs a coherent narrative compatible with it. The Operability loop sidesteps this opacity by construction. The artifact generated by the model, the verifier's deterministic output, and the diagnostic feedback that re-enters the context constitute an externally auditable trace of each step — not a verbal post-hoc rationalization, but the grounding process made inspectable. Within the configurations our experiments instantiate, the explanation of why a candidate was rejected is exactly the verifier's report; the explanation of why a candidate was accepted is the deterministic computation of the substrate. Where mainstream explainability methods add a post-hoc layer over an opaque core, Operability makes the closed-loop structure of inference itself the explanation. In this sense the framework belongs to the agenda of *Explainable Neurosymbolic AI*: not as an auxiliary justification module added on top of generation, but as a structural property of the inferential configuration in which generation occurs.

1.4 Research Questions

RQ1 (Taxonomy): Can Epistemic Hallucinations be isolated as a distinct consequence of how LLMs represent knowledge, qualitatively different from Structural errors and Training-induced biases, requiring a fundamentally different intervention strategy?
RQ2 (Syntax vs. Operability): Does formal syntax *alone* (e.g., programming languages) reduce epistemic hallucinations, or is the critical factor the closed verification loop with an external symbolic process?
RQ3 (Tool vs. Mechanism): Is the advantage observed in the executed conditions attributable to the use of an external tool, or specifically to the closed-loop mechanism through which the model encounters the consequences of its own generation? And under what conditions does this advantage extend across different KR formalisms?

1.5 Contributions

This paper makes three contributions.

(1) A mechanistic taxonomy of hallucinations that classifies failures by their architectural origin — Structural, Epistemic, and Training-induced — rather than by their observable surface, isolating Epistemic hallucinations as the category for which external symbolic verification is the appropriate intervention.

(2) The Operability framework, defined by three structural conditions — executability, semantic persistence, and interpretable feedback — under which a generative model can *epistemically appropriate* its own output through iterative interaction with a substrate that realizes those conditions. In the configurations we examine, the substrate is an external symbolic verifier; whether future architectures

might realize the same conditions through internal mechanisms remains an open empirical question, addressed by our cross-model probe (Experiment 3) and discussed in Section 4.3. Operability is framed as a property of the model–domain relation rather than of any individual tool: tool-augmented systems share some of the same functional ingredients (external process, returned output), but only configurations that jointly satisfy the three conditions in a closed loop yield epistemic appropriation rather than mere augmentation or evaluation.

(3) Cross-domain experimental evidence across two distinct modes of model–KR collaboration: verification (maze-solving with a Python interpreter) and computational delegation (job-shop scheduling with the Clingo ASP solver). The comparative design — mental versus executed conditions within each domain — isolates the closed-loop mechanism from the use of an external tool: the artifact-generation capacity is held constant across conditions, yet performance collapses when the loop is severed (56% vs. 100% for maze; 13% vs. 97% for ASP). The advantage is therefore attributable to the mechanism, not to the tool.

These contributions cohere around a single division of labor: the LLM provides probabilistic generation — candidate solutions through statistical pattern matching — while the external symbolic process provides deterministic grounding, whether through verification of the model's solution or through computational delegation of the problem itself.

The remainder of this paper is organized as follows. Section 2 reviews related work on hallucination taxonomies, the opacity of LLM inference, and neuro-symbolic approaches. Section 3 presents our causal taxonomy, the Operability framework, and the two experimental designs. Section 4 reports results. Section 5 answers the research questions and discusses limitations. Section 6 concludes.

2 Related Work

The dominant approach to classifying hallucinations operates at the level of observable outputs. Table 1 positions our taxonomy relative to the major existing surveys.

Taxonomy	**Basis**	**Categories**	**Guides interv.?**
Ji et al. (2023)	Task / output type	Intrinsic, Extrinsic (per NLG task)	No
Huang et al. (2025)	Output relation	Factuality, Faithfulness	Partially
Alansari and Luqman (2025)	Dev. lifecycle	6 pipeline stages	No
Rawte et al. (2023)	Source relation	Intrinsic, Extrinsic	No
Tonmoy et al. (2024)	Mitigation method	Grouped by technique	Inverted
Ours	**Architectural origin**	**Structural, Epistemic, Training-induced**	**Yes**

Table 1. Comparison with existing hallucination taxonomies. Prior surveys classify hallucinations by observable output (what the error looks like) or pipeline stage (when it arises). Our taxonomy classifies by *architectural origin* (why it occurs), yielding a distinct intervention strategy per category. A single factuality hallucination Huang et al. (2025) can be Structural (tokenization error), Epistemic (confabulation), or Training-induced (sycophancy)—each requiring a different remedy. Our taxonomy cuts *orthogonally* to prior classifications rather than replacing them.

Huang et al. (2025) also classify hallucination causes by pipeline stage: Data, Training, and Inference. Our taxonomy is fundamentally different in purpose and structure. It cuts *across* pipeline stages: tokenization errors occur during inference but are Structural, not Epistemic—they are mechanical failures, not knowledge failures. Epistemic errors involve knowledge management regardless of when they arise. Training-induced errors specifically target RLHF incentives that degrade veracity, distinct from other training problems. The purpose of our taxonomy is not lifecycle tracing but identifying the architectural point of intervention—and specifically isolating Epistemic hallucinations as the category for which no purely internal intervention is available.

Crucially, our taxonomy is not merely descriptive but *mechanistic*: it classifies hallucinations by their architectural origin—the form in which knowledge is represented and processed—so that each category admits a distinct class of intervention. For Structural errors, the natural intervention is mechanical (character-level access, error-correcting decoding, retrieval at the token level). For Training-induced errors, it is incentive-level (modifications to post-training procedures, calibration losses, decoupling reward from agreeableness). For Epistemic errors, the situation is qualitatively different: because the same parameters encode both verified facts and plausible interpolations, no internal mechanism—improved training, more careful prompting, or extended inference—can distinguish the two from within. External symbolic verification is therefore *necessary*, not merely useful.

This is the directional claim our experiments test: that Operability is *required* for Epistemic hallucinations, not that Operability is uniquely applicable to them. Where a Structural error happens to be reformulable as a verifiable artifact—a character-counting

task delegated to a Python interpreter, an arithmetic chain checked symbol-by-symbol—Operability may apply as well, but it is one option among several, and the mechanical intervention remains the more natural fix. What the taxonomy isolates is the category for which *no internal alternative exists*: Epistemic hallucinations cannot be resolved by any quantity of internal reasoning because the model lacks the architectural property—provenance tracking—that would be required to do so. The taxonomy thus functions as a diagnostic lens: given a hallucination, it identifies the architectural cause, distinguishes interventions that address the cause from interventions that treat symptoms, and—for Epistemic hallucinations—specifies the role of KR systems as the verification component the model architecturally lacks.

2.1 The Opacity of Inference and the Failure of Self-Correction

A convergent body of evidence demonstrates that LLMs cannot reliably inspect, explain, or correct their own reasoning processes. Turpin et al. (2023) show that Chain-of-Thought explanations can be plausible yet unfaithful: when generating a reasoning chain, the model is not describing its inference process (which remains unobservable) but constructing a plausible narrative post-hoc.

This opacity is compounded by systematic failures in self-correction. Stechly et al. (2025) demonstrate that self-verification in LLMs is fundamentally limited—verification is not easier than generation for approximate retrieval systems. Huang et al. (2024) show that intrinsic self-correction (without external feedback) fails; performance often *degrades* after self-correction attempts, as the model second-guesses correct answers. Madsen et al. (2024) find that self-explanations are convincing but often wrong, with faithfulness depending on explanation type, model, and task.

The underlying architectural limitation is the absence of provenance tracking. Tighidet et al. (2024) identify specific activations that correlate with parametric versus contextual knowledge selection, but models cannot access this distinction without external probing. Cheng et al. (2024) show that LLMs suppress parametric knowledge when contextual information is available, even when that context is irrelevant—they cannot intelligently integrate sources. Khalifa et al. (2024) demonstrate that standard training neglects attribution entirely, and explicit source-aware training is required for provenance tracking.

These findings converge on a structural conclusion: LLMs cannot internally distinguish verified knowledge from plausible interpolation. This is not a calibration problem amenable to improved training, but an architectural limitation. Since the model cannot self-verify, verification must be externalized.

A critical theoretical link connects these observations. Dherin et al. (2025) demonstrate that in-context learning is functionally equivalent to low-rank weight modification on the first MLP layer. This means that inference itself is a form of learning—each forward pass implicitly modifies the model's effective behavior based on context. Combined with Kalai and Vempala's result that hallucination is mathematically necessary during inference, this yields a clear implication: rather than demanding hallucination-free generation (which is provably impossible for calibrated models), the solution is creating

environments where hallucinations are harnessed through verified feedback that enters the context and refines subsequent behavior.

2.2 Neuro-Symbolic Approaches and KR for LLMs

The integration of neural and symbolic methods has been an active research programme for decades. d'Avila Garcez and Lamb (2023) characterise the current resurgence as the *third wave* of neurosymbolic AI, following symbolic AI and connectionism; Hitzler et al. (2022) survey the field along axes of representation, inference, and learning; De Raedt et al. (2020) trace the lineage from statistical relational learning. The dominant pattern across this tradition is *architectural integration*: symbolic structure is embedded into neural systems (logic tensor networks, differentiable theorem provers, knowledge graph embeddings) or reciprocally, neural learning is guided by symbolic priors. Kautz (2022) formalizes this as a classification of six types of neuro-symbolic systems (Type I–VI), distinguished by the tightness of the interface between the two components. Marcus (2020) argues, from a different angle, that robust AI requires hybrid architectures combining the strengths of both paradigms.

Our framework takes an orthogonal position. We do not propose an architectural integration: we do not modify the model, we do not embed symbolic structure into its weights, and we do not require any specific KR component. Instead, we identify the *structural conditions of the coupling* between a generative model and a deterministic substrate under which the resulting interaction yields epistemic appropriation rather than mere augmentation. The mechanism achieving those conditions may, as Figure 1 illustrates, be native to the domain, externally provided, or generated at runtime by the model itself. Where Kautz's Type III–VI describe integrations with progressively tighter neural-symbolic interfaces, Operability describes a different dimension entirely: the structural closure of a feedback loop with deterministic external semantics, independent of how that loop is architecturally realized.

This positioning has a direct consequence for the related work that motivates the present empirical study. Vasileiou and Yeoh (2025) demonstrate that combining SAT-based symbolic scheduling with LLM-generated natural language explanations achieves 100% explanation correctness, compared to 44–49% for LLM-only approaches—a result that resonates with our findings. Ren et al. (2025) benchmark 14 LLMs on Answer Set Programming tasks, showing that while LLMs handle simple entailment, they fail on answer set computation. These results empirically confirm the pattern our framework theorizes: LLMs generate plausible candidates, but symbolic KR systems are required for verification. Unlike specific hybrid systems such as TRACE-CS, however, our framework is grounded in the theoretical result that hallucination is mathematically unavoidable for calibrated models Kalai and Vempala (2024). The implication is not that any external tool grounds generation, but that grounding emerges when the configuration jointly satisfies the closed-loop conditions we identify in Section 3 — executability, semantic persistence, and interpretable feedback. Where these conditions are jointly met, KR systems function as the symbolic complement that probabilistic generation lacks.

3 Method and Framework

3.1 A Causal Taxonomy of Hallucinations

We propose a taxonomy that classifies hallucinations by their architectural origin, explicitly aimed at identifying points of intervention rather than cataloging observable phenomena. The taxonomy distinguishes three categories. We do not claim exhaustive coverage of all possible failure modes; the taxonomy is designed as a diagnostic tool oriented toward intervention, identifying the architectural origin of a hallucination in order to prescribe the appropriate class of remedy.

Structural Hallucinations arise from the computational process itself, independent of knowledge content. Tokenization errors occur because the model cannot perform operations requiring character-level access—tokenization obscures character boundaries, so counting letters in a word becomes unreliable not because the model lacks factual knowledge, but because it cannot "see" characters as discrete units Shin and Kaneko (2024). Reasoning chain errors occur when an early mistake in multi-step reasoning propagates through subsequent steps—the snowballing effect documented by Zhang et al. (2024)—including calculation errors, planning failures, and constraint forgetting where the model loses track of conditions imposed at the beginning of the task—that is, errors propagating mechanically through a reasoning chain, where a correct earlier step would have prevented the later failure. Crucially, each step appears locally valid, making these errors undetectable by the model itself but highly responsive to external feedback. These errors are *content-independent*: they arise from the structure of sequential computation, not from what the model knows or fails to know about the domain.

Epistemic Hallucinations arise from how the model manages knowledge, its sources, and its uncertainty. Pure confabulation generates completely invented but plausible factual information—the model produces outputs that are statistically coherent but ontologically empty, with no internal mechanism to distinguish verified knowledge from plausible interpolation Min et al. (2023). Faithfulness failures occur when the model ignores, distorts, or contradicts information explicitly provided in the context—pre-trained knowledge outweighs provided context, or the model adds information not verifiable from the source Huang et al. (2025). Temporal hallucinations arise when models lack mechanisms to track knowledge currency, producing outdated or anachronistic information Vu et al. (2023), or conflate events across different periods due to weak temporal representation in training data.

Training-induced Hallucinations are behaviors explicitly incentivized by post-training procedures that degrade factual accuracy. Sycophancy occurs when the model modifies a correct response to align with user feedback, even when the user is wrong—driven not by uncertainty but by RLHF reward models that favor agreeableness Sharma et al. (2024). Completion bias arises when the pressure to generate a complete response prevails over verification, again incentivized by reward models that prefer confident and complete answers.

The critical distinction is that Epistemic hallucinations are qualitatively different from the other two categories. Unlike Structural errors (which are mechanical—the model cannot count characters because tokenization obscures boundaries) and

Training-induced errors (which are incentive-driven—the model agrees because RLHF rewards agreeableness), Epistemic hallucinations arise from a fundamental architectural limitation: the same parameters encode both verified facts and plausible interpolations, making them indistinguishable at inference time. The model does not "know" whether it knows or interpolates. Since the provenance of knowledge is architecturally untrackable, external verification is not merely useful but *necessary*—it is the only mechanism that can distinguish truth from plausible confabulation.

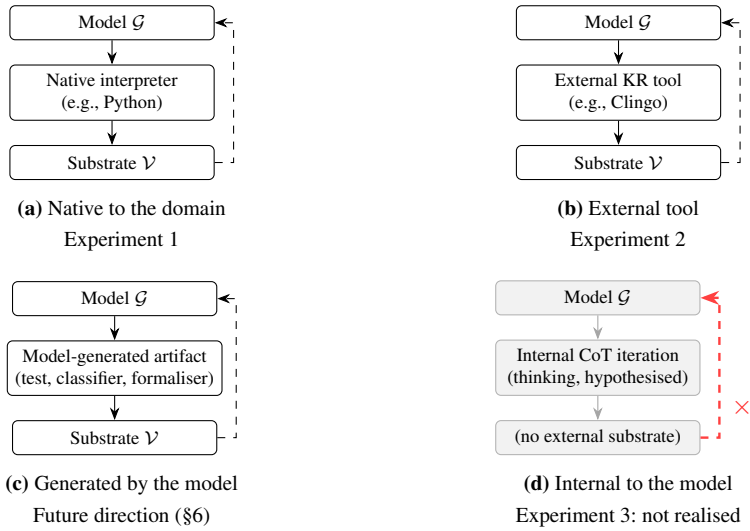
3.2 The Operability Framework

Definition 1. Operability. Operability *is a property of the model–mechanism–domain configuration: a configuration admits Operability when (i) the model can produce artifacts in the domain, (ii) the configuration includes a mechanism — whether native to the domain, provided as an external tool, generated by the model itself, or, in principle, realized internally — that satisfies executability, semantic persistence, and interpretable feedback, and (iii) these conditions are realized in a closed loop through which the artifact's behavioral consequences re-enter the model's inferential context.*

A model–mechanism–domain configuration exhibits *Operability* when three conditions are jointly met:

1. **Executability**: there exists an external oracle that deterministically evaluates the artifact and establishes whether it "works."
2. **Semantic Persistence**: the artifact preserves its meaning across iterations, enabling cumulative refinement—each correction builds on the previous state rather than starting from scratch.
3. **Interpretable Feedback**: the oracle's output is structured in a form that the model can process and act upon for correction—error messages, test results, constraint violations.

More precisely, the Operability loop iterates through generation, verification, and appropriation. At step i, the generator $\mathcal{G}$ produces artifact α_i conditioned on the cumulative context C_i (comprising the original prompt, all previously generated artifacts $\alpha_1, \ldots, \alpha_{i-1}$, and all prior feedback $\mathcal{F}_1, \ldots, \mathcal{F}_{i-1}$). The verifier $\mathcal{V}$ evaluates α_i yielding feedback $\mathcal{F}_i$, and the next candidate α_{i+1} is conditioned on $C_i \cup \{\mathcal{F}_i\}$. Through semantic persistence, the model encounters the consequences of its own generation: each $\mathcal{F}_i$ acts as a hard constraint that prunes the probabilistic search space, forcing convergence toward valid outputs. Operability is therefore a property whose third condition is itself dynamic: it holds of a configuration through the convergence the closed loop enacts. The theoretical foundation of this framework connects two results. Dherin et al. (2025) demonstrate that in-context learning operates through an implicit mathematical mechanism: when a transformer processes context, the self-attention layer combined with the MLP produces a low-rank weight update. This means that when verified feedback from an external oracle enters the context window, it functions as a genuine learning signal, implicitly modifying the model's effective behavior for subsequent generation within that interaction. Combined with Kalai and Vempala's proof that hallucination is



(a) Native to the domain
Experiment 1

(b) External tool
Experiment 2

(c) Generated by the model
Future direction (§6)

(d) Internal to the model
Experiment 3: not realised

Figure 1. Four instantiations of the mechanism in the Operability triad (Definition 1). Each panel shows the same triadic structure *(model, mechanism, substrate)* closed by feedback. What varies across panels is the *origin* of the mechanism that satisfies the three structural conditions: (a) native to the domain, as in Experiment 1, where the Python interpreter is the deterministic substrate; (b) an external KR tool invoked by the model, as in Experiment 2 with Clingo; (c) an artifact generated by the model itself at runtime (e.g., a test harness, a classifier, a formal encoding produced *ad hoc*), opening Operability to domains where no native substrate is available; (d) in principle, an internal closed loop within the model's own inference — which Experiment 3 (§4.3) shows is *not* achieved by any of the thinking-augmented models tested. The framework is silent on which form realises the conditions; what matters is the conjunction.

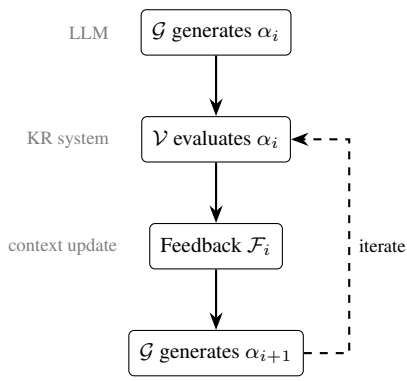


Figure 2. The Operability loop. $\mathcal{G}$ produces artifact α_i; the verifier $\mathcal{V}$ evaluates it and returns feedback $\mathcal{F}_i$, which enters the context for the next generation cycle.

mathematically necessary during any single inference, Operability enables cumulative correction across multiple inferences: each feedback instance enters the context and, per Dherin et al.'s result, functions as an implicit weight modification signal for subsequent generation. This involves two complementary levels of learning. At the first level (intra-inferential), implicit weight modification occurs within a single forward pass Dherin et al. (2025). At the second level (inter-inferential), an agentic loop operates: $\mathcal{G}$ generates α_i, $\mathcal{V}$ evaluates it, and $\mathcal{F}_i$ enters the context. Operability exploits the first level to enable the second: by reintroducing $\mathcal{F}_i$ into the context, the verification loop transforms transient implicit updates into a cumulative learning signal grounded in symbolic truth. The connection to Kalai and Vempala (2024) is direct: if hallucination is necessary during any single inference (Level 1), the solution is not to prevent hallucination but to create environments where hallucinations are identified and corrected through the agentic loop (Level 2). In operable domains, hallucination becomes productive exploration—a generative process that is iteratively grounded against symbolic truth. In non-operable domains, the same hallucination renders outputs epistemically opaque, because no mechanism exists to distinguish generation from confabulation.

Operability identifies when tool-use enables epistemic appropriation. Tool-augmented LLM systems are now a broad family — tool-use Schick et al. (2023), program-aided reasoning Gao et al. (2023), agentic frameworks Yao et al. (2023) — and they share an obvious functional property: an external process is invoked, and its output enters the model's reasoning. What this family lacks, however, is a principled account of which configurations of external interaction yield epistemic grounding, and which yield only informational augmentation. Operability fills this gap. The framework identifies, within tool-augmented systems, a structural condition within tool-augmented systems: when an external process satisfies executability, semantic persistence, and interpretable feedback in a closed loop, the resulting interaction is no longer merely tool-use but epistemic appropriation. We distinguish three levels of external interaction (Table 2); only the third meets these conditions jointly.

Tool-use provides information the model lacks: a search engine retrieves documents, a calculator returns a numerical result, an API furnishes current data. The external process augments the model's knowledge but does not evaluate the model's own output. The model *receives*; it does not *produce and then observe*.

Execute-and-check (as in single-shot program-aided reasoning Gao et al. (2023)) goes further: the model generates an artifact, an external process evaluates it, and the result is returned. However, without iterative feedback on the same persistent artifact, the model cannot integrate, refine, or learn from its errors within the interaction. A single execution returns a verdict, not a learning opportunity.

Operability requires all three conditions — executability, semantic persistence, interpretable feedback — operating in a closed loop. The previous two levels share some of these properties: tool-use can have executability and interpretable feedback but lacks iteration on a persistent artifact; execute-and-check has executability and persistence but typically lacks the iterative loop. Only Operability has all three jointly, and it is this conjunction that yields the epistemic difference.

	Tool-use	Execute-and-check	Operability
LLM output verified?	No	Partially	Yes
Feedback type	Informative (new data)	Result (pass/fail)	Diagnostic (what failed)
Iteration on artifact?	No	Rarely	Required
Semantic persistence?	N/A	Sometimes	Required
Epistemic relation	Augmentation	Evaluation	*Appropriation*
Example	RAG, Toolformer	PAL (single-shot)	This framework

Table 2. Three levels of external interaction in tool-augmented LLM systems. All three involve calling external processes; what differs is the epistemic relation between the model and what the external process produces. Tool-use augments the model's knowledge with information it lacks; execute-and-check evaluates a single output; Operability identifies the structural conditions under which iterative external interaction yields *epistemic appropriation* rather than mere augmentation or evaluation.

Epistemic appropriation through externalization. The structural difference identified above has a precise epistemic counterpart. In tool-use, the external process provides information absent from the model's parameters; the model's epistemic state is *augmented*. In execute-and-check, a single output is evaluated; the model's epistemic state is *assessed*. In Operability, the model *externalizes its internal state* into a manipulable artifact whose behavioral consequences — failures, constraint violations, error traces — reveal properties of the model's "reasoning" that are architecturally inaccessible through introspection Turpin et al. (2023). Through iterative production, observation, and integration, the model's effective behavior is progressively shaped by the consequences of its own generation — a mechanism we term *epistemic appropriation*. The artifact is not a tool the model uses but a medium through which it encounters its own inference in externalized form. Building on the result that in-context processing has been shown to be functionally equivalent to weight modification Dherin et al. (2025), we conjecture that iterative feedback within a closed loop functions analogously, for a statistical system, to the causal grounding that distinguishes *owned* knowledge from received information.

3.3 Programming as a Paradigmatic Case

Programming serves as a paradigmatic—not unique—instance of Operability. A compiler or interpreter provides deterministic evaluation (Executability). Code is an artifact with explicit state, variables, and structure that persists across modification cycles (Semantic Persistence). Error messages, stack traces, and test results are structured outputs that the model can parse and act upon (Interpretable Feedback).

Crucially, formal syntax alone—without execution—does not provide these properties. A model can write syntactically correct code with recognized patterns and clear

semantics, and still produce wrong results. Syntax enables Operability (it makes execution possible) but does not *constitute* Operability. This distinction is central to our experimental design and, as we will show, is confirmed empirically.

The abstract properties that make programming operable are not exclusive to programming. We return to this generalization in Section 5, after presenting our experimental results.

3.4 Experiment 1: Maze-Solving

We designed a task requires maintaining state over time, respecting sequential dependencies, and multi-step planning where greedy strategies fail. The task is maze-solving with keys and doors: the solver navigates a grid from start to goal, but must collect keys (lowercase letters) before passing through corresponding doors (uppercase letters). Each state is a triple (r, c, S) where (r, c) is the current cell and $S \subseteq \{1, \ldots, k\}$ is the set of keys collected so far. The same cell visited with different key sets constitutes a different state, yielding a state space of $O(n^2 \times 2^k)$—exponential in k, the number of keys.

Mazes were generated procedurally with horizontal zones separated by doors, guaranteeing all keys are required. Greedy pathfinding toward the goal fails because the solver must sometimes move away from the goal to collect a key required for a door along the shortest path. This forces the model to maintain and update an internal representation of state—precisely the kind of reasoning where probabilistic interpolation systematically fails.

We tested three conditions, using identical maze representations in Python matrix format, to avoid input confounds: **NL** (Natural Language): the model reasons in prose and outputs a path as a list of coordinates.
CODE_mental: the model writes a BFS algorithm with state tracking, then is asked to mentally deduce the path the algorithm would produce—without executing it.
CODE_executed: the model writes the same type of algorithm, which is then executed by a Python interpreter, with output returned to the model. These conditions test two hypotheses. **H1** (Syntax Effect): if formal syntax aids reasoning, CODE_mental should perform at least as well as NL. **H2** (Execution Effect): if the feedback loop matters, CODE_executed should outperform CODE_mental.

The protocol was applied to two non-thinking models at the same three difficulty levels — Easy (8×8 grids, 1 key, state space $\sim$128), Medium (10×10 grids, 2 keys, state space $\sim$400), and Hard (12×12 grids, 3 keys, state space $\sim$1152) — with 15 trials per cell, 45 per condition, and 135 per model. The primary experiment used Claude 3.5 Sonnet. The same protocol was replicated on Gemini 2.5 Flash, a non-thinking model from a different lineage, to test whether the effect is artefactual of Sonnet 3.5's specific post-training regime. Paths were validated by simulated traversal: checking that the path starts at the origin and ends at the goal,that each step moves to an adjacent cell (Manhattan distance = 1), that no step traverses a wall, and that doors are only passed when the corresponding key has been previously collected.

3.5 Experiment 2: ASP Scheduling (Cross-Domain Validation)

To test whether Operability extends beyond programming to a genuine KR formalism, we replicated the three-condition design using job-shop scheduling with Clingo as the symbolic verifier. N jobs must be scheduled on M machines respecting precedence constraints, no-overlap constraints, and a tight deadline equal to the optimal makespan (zero slack). The state space grows combinatorially with job count and machine contention, and greedy strategies fail when precedence chains force non-obvious orderings—paralleling the key-before-door constraint in the maze task. Three conditions mirror the maze experiment:
NL: reason in prose (single attempt, no feedback).
ASP_mental: write Clingo rules, mentally deduce the answer set (single attempt, no feedback).
ASP_executed: write Clingo rules, executed by the solver with feedback loop (up to 3 attempts).

Critically, NL and ASP_mental receive no feedback—only ASP_executed enters the Operability loop. This isolates the effect of the symbolic verification cycle. Cross-model replication of the ASP design was not pursued: this experiment serves the cross-domain question (does Operability transfer beyond Python to a genuine KR formalism), while cross-model replication is conducted on the maze task (Section 3.6), which admits richer thinking-trace analysis. Experiments used Claude 3.5 Sonnet at two difficulty levels: Medium (10 jobs, 3 machines, 10 precedences; 5 trials) and Hard (14 jobs, 4 machines, 14 precedences; 25 trials), for 30 trials per condition and 90 trials overall.

3.6 Experiment 3: The Operability Signature on Thinking-Augmented Models

Experiments 1 and 2 establish the Operability effect on non-thinking models, measured by accuracy. The emergence of *thinking-augmented* architectures — in which the model generates substantial intermediate reasoning before producing its final answer — raises a distinct theoretical question. Such models conduct extensive self-referential computation that resembles, at the surface, the iterative refinement that Operability requires from an external loop. If the effect persists despite this internal scaffolding, the persistence itself constitutes evidence that no amount of intra-model reasoning substitutes for externally closed verification. If instead the effect disappears, the framework requires revision: Operability would emerge as a property eventually internalized by sufficient inference budget. Experiment 3 probes this question on two thinking-augmented models, Gemini 3 Flash Preview and Claude Sonnet 4.6.

We attempted to run both models on the protocol of Experiment 1; accuracy saturated across all three conditions for both. We then iterated on the maze configuration — increasing grid size, key count, and adversarial filtering — looking for a setup that would discriminate the conditions on accuracy. No single configuration achieved this for both models simultaneously: configurations where Gemini 3 Flash showed accuracy variance were configurations where Sonnet 4.6 failed to emit any output within bounded inference resources, and vice versa. This forced us to seek a sweet spot — the minimum

difficulty at which both models produced meaningful output without fully saturating the task — which we settled on at 12×12 with 4 keys and adversarial filtering. The maze task is the natural locus for this shift: its CoT traces are spatially structured and admit fine-grained diagnostic reading in a way that ASP traces, dominated by constraint syntax, do not. Path efficiency is defined as the ratio of the model's path length to the BFS-optimal length on the same instance. An efficiency of 1.000 indicates that the model's reported path coincides with the deterministic shortest-path output. Values of CODE_mental strictly greater than 1.000 indicate that the model reaches the goal validly, but along a trajectory that its own written algorithm, if executed, would not produce. Efficiency thus operationalizes a notion of *trace fidelity*: the extent to which the model's reported solution matches the deterministic behavior of the formal artifact it claims to be following. Both models were tested at 10 trials per condition (30 total per model). The small sample is a methodological consequence rather than a constraint: with accuracy saturating, formal hypothesis testing on accuracy is uninformative, and the locus of evidence shifts to aggregate efficiency together with close-reading of individual reasoning traces (Section 4.4).

4 Results

4.1 Experiment 1: Maze on Non-Thinking Models

Condition	**Easy** (8×8, 1 key)	**Medium** (10×10, 2 keys)	**Hard** (12×12, 3 keys)
NL	87%	93%	33%
CODE_mental	100%	40%	27%
CODE_executed	100%	100%	100%
Overall (45 trials/cond.):			
NL		71%	
CODE_mental		56%	
CODE_executed		100%	

Table 3. Maze accuracy on Claude 3.5 Sonnet by condition and difficulty level (15 trials per cell, 45 per condition, 135 total). CODE_executed achieves perfect accuracy; CODE_mental performs worse than natural language.

Sonnet 3.5: quantitative analysis. Statistical analysis confirms the significance of these patterns. For H1 (CODE_mental vs. NL), the odds ratio is 1.97 in favor of NL—natural language reasoning actually outperforms writing code without execution. Fisher's exact test yields $p = 0.189$, which is not statistically significant at $\alpha = 0.05$, but the directional trend is against the hypothesis that syntax helps. H1 is not supported: formal syntax does not improve reasoning performance.

For H2 (CODE_executed vs. CODE_mental), the contrast is definitive. Fisher's exact test yields $p < 0.0001$—the improvement from 56% to 100% is highly significant. H2 is strongly confirmed: the interpreter feedback loop dramatically transforms performance.

The 100% accuracy of the executed condition, compared to the 56% of the identical formal language *without* execution, demonstrates that the entire advantage lies in the verification loop, not in the syntactic properties of code.

Sonnet 3.5: error analysis. The distribution of error types across conditions reveals qualitatively different failure modes. Wall traversal errors and non-adjacency errors are common in both NL and CODE_mental but absent or quickly corrected in CODE_executed. Notably, door-without-key errors—passing through a locked door without the corresponding key—are rare in NL but common in CODE_mental. This suggests that translation into code does not improve the model's ability to track state; if anything, it degrades it.

The most revealing finding concerns the CODE_mental condition specifically. The model frequently writes a *correct* BFS algorithm—but then produces an *incorrect* path when asked to mentally trace the algorithm's execution. It writes code that would solve the maze correctly if executed, yet when asked what that code would produce, it gives a wrong answer.

This dissociation is deeply informative. Whatever internal structure the model has learned about BFS, it cannot deterministically apply it to this specific input.[†] The interpreter provides exactly what the model lacks: not a representation of BFS, but its deterministic execution on this specific input. This dissociation clarifies the link between our experimental task and Epistemic hallucination. The model's failures are not failures of reasoning ability—the CODE_mental condition demonstrates that the model *can* write correct algorithms. They are failures of epistemic self-tracking: the model cannot verify whether its own intermediate states (keys collected, constraints satisfied) reflect actual computation or plausible interpolation. This is not a reasoning deficit but a provenance deficit—the model does not know what it knows about its own trajectory.

The error patterns across conditions tell a coherent story. In the NL condition, the model loses state tracking as complexity increases—it "forgets" which keys have been collected. In the CODE_mental condition, the overhead of translating the problem into code, mentally simulating execution, and translating back to a path introduces additional error points without any compensating benefit. In the CODE_executed condition, initial errors are systematically identified by the interpreter and corrected through the feedback loop—the cycle of generation, evaluation, and appropriation that Operability enables.

Replication on Gemini 2.5 Flash. The same protocol applied to Gemini 2.5 Flash, a non-thinking model from a different lineage and training regime, reproduces the directional pattern (Table 4): CODE_mental (60%) underperforms NL (75%), and CODE_executed reaches 100%. The 15-percentage-point gap between NL and

[†]Whether large language models develop implicit world representations Li et al. (2023); Gurnee and Tegmark (2024), or whether such representations are fundamentally limited or absent Kang et al. (2024), is an active debate beyond the scope of this work. Our argument is orthogonal: even granting that such representations exist, our experimental result shows that they remain epistemically opaque to the model itself — it cannot verify, from within, whether the trajectory it produces reflects deterministic execution of those representations or plausible interpolation across similar trajectories seen in training. Operability addresses this opacity regardless of which side of the debate prevails.

CODE_mental closely tracks the corresponding 15-point gap on Sonnet 3.5 (71% vs. 56%). The result rules out the most parsimonious alternative explanation of the original finding — that Sonnet 3.5's specific post-training induced an artefactual disadvantage in CODE_mental — and supports the claim that the effect tracks an architectural feature of the model class rather than a peculiarity of one training regime.

Condition	**Easy**	**Medium**	**Hard**
NL	100%	71%	50%
CODE_mental	100%	43%	33%
CODE_executed	100%	100%	100%
Overall: NL 75%, CODE_mental 60%, CODE_executed 100%			

Table 4. Maze accuracy on Gemini 2.5 Flash, original protocol (15 trials per cell, 45 per condition, 135 total). The directional pattern of Sonnet 3.5 reproduces on a non-thinking model from a different lineage.

4.2 Experiment 2: ASP Cross-Domain Validation (Sonnet 3.5)

Condition	**Medium**	**Hard**	**Overall**
NL	40%	20%	23%
ASP_mental	40%	8%	13%
ASP_executed	100%	96%	97%

Table 5. ASP scheduling accuracy on Claude 3.5 Sonnet (5 medium + 25 hard trials per condition, 90 total). Fisher's exact test for ASP_executed vs. ASP_mental: $p < 0.001$.

The Operability pattern replicates across domain and formalism (Table 5). ASP_executed (97%) substantially outperforms ASP_mental (13%) and NL (23%), with the execution effect highly significant ($p < 0.001$). ASP_mental performs worse than NL, replicating the directional pattern observed in the maze experiment, though the comparison is not statistically significant in either case ($p = 0.506$; cf. $p = 0.189$ for the maze).

Controlling for retry advantage. A potential confound is that ASP_executed receives up to 3 attempts while NL and ASP_mental are single-shot. However, the multiple attempts *are* the Operability mechanism, not a confound: without diagnostic feedback, additional attempts amount to uninformed repetition—the model does not know *what* to correct. The first-attempt data—the success rate of the initial Clingo encoding submitted before any feedback-driven correction—confirms this: ASP_executed already succeeds in 47% of trials on its first submission—double the NL rate (23%) and triple the ASP_mental rate (13%). The gap at first attempt reflects the fact that Clingo *solves* the problem when given a correct encoding, whereas NL and ASP_mental require the model itself to compute the solution. Subsequent attempts then amplify this advantage through targeted correction: the feedback loop raises success from 47% to 97% precisely because each error message identifies a specific constraint violation, enabling directed repair rather than blind retry.

The ASP experiment reveals a distinct mode of Operability. In the maze, the LLM writes an algorithm that the interpreter executes — the model is the author of the solution. In ASP, the LLM encodes constraints that Clingo solves — the model is the formalizer, and the KR system is the solver. Both require the same three conditions (executability, semantic persistence, interpretable feedback), but the division of labor differs: verification in one case, computational delegation in the other. This suggests Operability admits at least two distinct modes of model–KR division of labor, and may extend to further configurations beyond pure verification.

Symbolic Bottleneck. The ASP experiment provides direct evidence for the Symbolic Bottleneck discussed in our limitations. Only 47% of ASP code was valid on the first attempt, compared to approximately 80% for Python in the maze experiment. Crucially, all 19 first-attempt failures were *semantic* (incomplete overlap constraints), not syntactic—the model can write valid ASP syntax but fails to encode the full constraint semantics. Despite this, 97% of trials succeeded within 3 attempts: the feedback loop compensates for the bottleneck, but requires more iterations than with a high-resource formalism.

Divergent failure modes. The error distributions of the two non-executed conditions are qualitatively distinct. NL failures are predominantly state-tracking omissions: in 13 of 23 failures the model omits one or more jobs from the schedule entirely, as if it "forgot" they existed—a classic provenance-loss pattern. ASP_mental failures, by contrast, are formalization errors: 14 of 26 failures involve machine-overlap violations caused by incomplete constraint encodings. The model writes syntactically valid ASP that captures precedence and deadline constraints but under-specifies the no-overlap condition—it produces a partial formalization that *looks* complete. Both failure modes are resolved by the feedback loop. In ASP_executed, Clingo's output identifies the specific violated constraint, enabling targeted repair of the encoding; in CODE_executed, the interpreter's runtime feedback (wall traversal, missing key) identifies the specific point at which the candidate path violates a state constraint, enabling analogous targeted correction. This divergence is consistent with the taxonomy: NL errors reflect the model's inability to track its own intermediate state (Epistemic), while ASP_mental errors reflect its inability to verify the completeness of its own formalization — a second facet of the same provenance limitation, now operating at the level of constraint specification rather than solution computation.

4.3 Experiment 3: Thinking-Augmented Models

On the two thinking-augmented models, accuracy alone becomes uninformative — not because the underlying provenance failure has disappeared, but because its surface manifestation has migrated. The two models exhibit qualitatively distinct failure modes; together they delineate the boundary within which the Operability signature persists despite extensive internal reasoning.

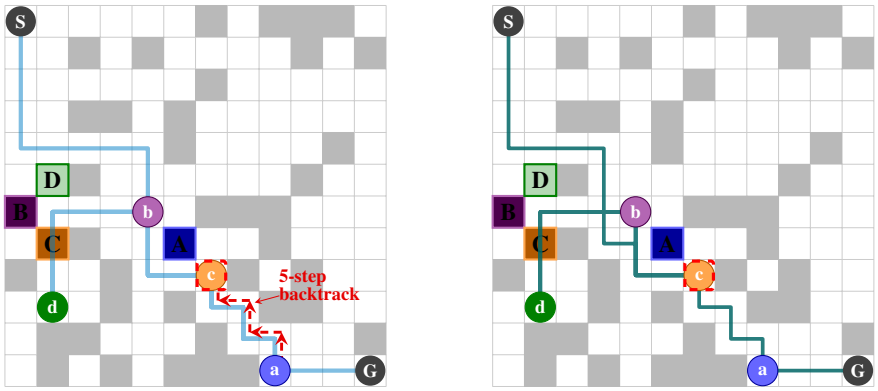
Gemini 3 Flash: efficiency migration and unfaithful trace. Gemini 3 Flash achieves 100% accuracy across all three conditions on the harder protocol (Table 6), yet path

efficiency reveals exactly the gradient that the original accuracy hierarchy predicted: NL produces valid but circuitous paths (1.301 average), CODE_mental produces nearly-optimal paths (1.042), and CODE_executed produces strictly optimal paths (1.000). The direction NL $>$ CODE_mental $>$ CODE_executed in deviation from optimal exactly mirrors the direction of the original error-rate hierarchy on Sonnet 3.5. The phenomenon has not been eliminated; it has been re-encoded in a metric that the binary accuracy criterion cannot resolve.

Condition	**Accuracy**	**Path efficiency**
NL	100% (10/10)	1.301
CODE_mental	100% (10/10)	1.042
CODE_executed	100% (10/10)	1.000

Table 6. Maze results on Gemini 3 Flash Preview, harder protocol (10 trials per condition, 12×12 with 4 keys, adversarial filtering). Accuracy saturates; the Operability signature migrates to path efficiency.

The efficiency gradient does not reflect internalization of the verification loop. Direct inspection of the chain-of-thought shows that the model writes a correct BFS algorithm and *separately* solves the maze through a parallel, independent process — heuristic waypoint decomposition organized into "Phase 1, Phase 2, …" segments. The two processes do not communicate: the reported path is the output of the heuristic, not of the algorithm. The phenomenon Turpin et al. (2023) document for short-prompt CoT — chains of thought that are plausible but unfaithful to the computation they ostensibly describe — here scales to the extended-thinking regime. The CoT verbalizes one algorithm and executes another. Figure 2 reports a case in which this dissociation is directly observable.



(a) Reported path: 52 steps, efficiency 1.233 **(b)** BFS-optimal path: 42 steps, efficiency 1.000

Figure 3. Trace-fidelity failure on Gemini 3 Flash, seed 5003000 (referenced in Box 1). The model's written BFS algorithm collects keys automatically upon cell entry; the path narrated by the model in its CoT (panel a) passes through cell (8,6) at step 14 during Phase 2, yet at Phase 3 the narration reports a 5-step backtrack to "collect" key 'c' from that same cell. The cell is visited three times in total. Panel (b) shows the path that a deterministic execution of the model's own code would produce: 42 steps, no redundant backtracks, key 'c' acquired implicitly at step 14. The red highlight marks (8,6); doors A, B, C, D open with their lowercase keys.

A faithful execution of the model's own written code would have key 'c' in `new_keys` at the end of Phase 2; the 5-step backtrack contradicts the semantics of the artifact the model produced. The reported path is valid (it solves the maze) but is not the path the written code would generate. The model's CoT verbalizes an algorithm (BFS) while reporting the output of a different process (heuristic decomposition), and the discrepancy is undetectable from within: this is the trace-fidelity failure that the path-efficiency metric operationalizes, and a direct instance of the unfaithful-CoT pattern Turpin et al. (2023) now manifest in the extended-thinking regime.

Sonnet 4.6: substrate exhaustion. Sonnet 4.6 exhibits a qualitatively distinct failure mode (Table 7). On CODE_mental, the model fails on 10/10 trials — not by producing wrong answers, but by failing to produce any answer within bounded resources. All CODE_mental failures are output-token exhaustion events (default 32,000-token budget) after extended-thinking iterations. Two of the NL failures show the same failure mode, suggesting that the substrate-exhaustion regime is not specific to CODE_mental for this model but is amplified by it.

Condition	Accuracy	Path efficiency
NL	80% (8/10)[†]	1.133
CODE_mental	0% (0/10)[‡]	—
CODE_executed	100% (10/10)	1.000

Table 7. Maze results on Claude Sonnet 4.6, harder protocol. [†]2/10 NL failures are output-token exhaustion events; the remaining 8 produced valid paths, and the reported efficiency averages over those 8 successful trials. [‡]All 10 CODE_mental failures are output-token exhaustion events with no path emitted; efficiency is undefined.

A single CODE_mental trial that we allowed to run beyond the default budget illustrates the failure mode (Box 2); the full stream log is included in the accompanying repository.

Box 2. Substrate-exhaustion failure on Sonnet 4.6.
On one CODE_mental trial allowed to exceed the default budget, the model emitted four consecutive extended-thinking blocks totalizing ≈127,000 output tokens over 27 minutes of wall-clock time, at a cost of $1.97, before hitting the 32,000-token per-iteration ceiling on the fifth attempt. Each block opens with the declared intention to write a BFS solution and mentally trace it (*"Let me analyze this maze and write a BFS solution, then mentally trace through it"*). No block ever produces the BFS code. Instead, each block engages in increasingly elaborate manual reasoning: pairwise Manhattan distances between key locations, lower bounds on tour length, alternative collection orderings, partial verification of specific transitions through doors. The model never internally simulates its own (unwritten) BFS; it solves a TSP-like relaxation manually, repeatedly, without converging. This is a complementary form of the Turpin et al. (2023) pattern: not only does the verbalized chain-of-thought falsifies the underlying computation (BFS is declared, manual distance calculation is executed), the model cannot align even its own stated plan with what it then does. The failure mode is the dual of Box 1: where Gemini 3 Flash accepts a heuristic shortcut and emits a fictional trace of a written artifact, Sonnet 4.6 refuses the shortcut, commits to faithful manual computation, and exhausts its substrate before emitting any artifact at all.

The metric of failure is capability-dependent; the architectural cause is not. The two failure modes share an architectural root. Neither Gemini 3 Flash nor Sonnet 4.6 can deterministically execute a formal artifact within bounded inference resources. Gemini 3 Flash resolves the intractability by accepting a heuristic shortcut and emits a narrative trace that is decoupled from the written artifact's actual semantics. Sonnet 4.6 refuses the shortcut and exhausts its generation budget. Both manifest, in different surface forms, the same impossibility: *the model cannot serve as its own deterministic verifier*, even when it possesses the formal capacity to write a correct verifier in code. This is the structural fact that Operability addresses by externalizing the verification loop, and that no degree of extended thinking can eliminate from within.

Across the four models tested across Experiments 1 and 3, the Operability signature appears in three different metrics — accuracy on Sonnet 3.5 and Gemini 2.5 Flash, path efficiency on Gemini 3 Flash, output-token exhaustion on Sonnet 4.6 — but the direction is invariant: the condition with the closed external loop (CODE_executed)

achieves deterministic fidelity, and no internal process in any model tested achieves it. **The provenance failure is architecturally invariant; the metric of its visibility is capability-dependent.**

5 Discussion and Limitations

5.1 Answering the Research Questions

RQ1: The Taxonomy as Diagnostic Framework. Our taxonomy yields a specific experimental implication: Epistemic hallucinations — arising from untraceable knowledge provenance — should respond selectively to external symbolic verification, while remaining unaffected by syntactic structure alone. The maze task triggers primarily Epistemic hallucinations, though the classification requires care. The model's errors are not mechanical failures in reasoning chains (Structural) — each step is locally coherent. Nor are they incentive-driven distortions (Training-induced). Rather, the model cannot distinguish verified state (a key was collected at step n because the path passed through cell (r, c) containing that key) from interpolated state (it *seems plausible* that the key was collected, based on pattern similarity with correct solutions). This is a provenance failure: the model's parameters encode both the pattern of correct maze-solving and the pattern of plausible-sounding paths, and it cannot distinguish between them at inference time. The classification as Epistemic rests on this distinction — the failure is not in the computation but in knowing the origin of one's own intermediate states.

The intervention our taxonomy associates with Epistemic hallucinations — external symbolic verification — eliminates these errors entirely (100% accuracy with the interpreter), while formal syntax without verification fails (56%, worse than natural language at 71%). This differential response is consistent with the taxonomy's prediction: the same errors that resist syntactic intervention yield completely to symbolic verification. The dissociation between correct algorithm and incorrect path execution further rules out a Structural interpretation: structural decay would corrupt both the algorithm and its trace, but here the algorithm is intact and only the trace fails. This isolates the failure to provenance — knowing what one's own intermediate states are — rather than to computation per se.

The taxonomy does not predict that Operability fails for Structural or Training-induced errors; it predicts that Operability is *necessary* for Epistemic hallucinations because no internal alternative exists. Whether Operability also applies to Structural errors that happen to be reformulable as verifiable artifacts (e.g., counting tasks via a Python interpreter, arithmetic chains via symbolic checking) is an interesting empirical question, but it is orthogonal to the necessity claim and we do not address it here. The ASP experiment offers indirect support: the divergent failure modes observed between NL (state-tracking omissions) and ASP_mental (incomplete constraint formalizations) are consistent with a provenance problem operating at two levels — at the level of solution computation and at the level of formalization completeness — rather than with mechanical error propagation. This is the pattern the taxonomy predicts for Epistemic hallucinations, distinct from the snowballing dynamics of Structural errors. Experiment 3

strengthens this reading: Epistemic hallucinations are not specific to one generation of LLMs but persist across model capability. The manifestation migrates — from accuracy degradation on Sonnet 3.5 and Gemini 2.5 Flash to path-efficiency degradation on Gemini 3 Flash to substrate exhaustion on Sonnet 4.6 — but the underlying provenance failure is invariant. Box 1 reports the most direct evidence: Gemini 3 Flash, a thinking-augmented model that achieves 100% maze accuracy, produces a narrative trace that contradicts the semantics of the BFS algorithm it has just written, and cannot detect the contradiction from within. Extended thinking does not give the model access to the provenance of its own intermediate states; it only enables more elaborate confabulation about them. The taxonomy's prediction — that only external symbolic verification can ground these states — is therefore an architectural prediction, not a capability-dependent one.

RQ2: Syntax alone is not sufficient. The hypothesis that formal syntax reduces epistemic hallucinations independently of execution is not supported on non-thinking models: CODE_mental underperforms NL on Sonnet 3.5 (56% vs. 71%, $p = 0.189$), on Gemini 2.5 Flash (60% vs. 75%), and the analogous comparison on ASP for Sonnet 3.5 (13% vs. 23%, $p = 0.506$). None of the individual comparisons reaches statistical significance, but the directional consistency across two model families and two formalisms supports the reading that writing code without execution adds translation overhead — problem $\rightarrow$ code $\rightarrow$ mental simulation $\rightarrow$ path — that creates additional points of failure. For a pattern-matching system, code without execution is, plausibly, "harder prose": it introduces formal constraints that the model can satisfy syntactically but cannot leverage computationally.

On thinking-augmented models, the picture is finer-grained. With accuracy saturated, the comparison migrates to efficiency, where Gemini 3 Flash's CODE_mental produces nearly-optimal paths (1.042) while NL produces longer ones (1.301). The written artifact does provide scaffolding — it constrains the model's heuristic reasoning into a more rigorous waypoint decomposition than NL alone elicits — but the scaffolding is bounded, and never closes the gap to executed code (1.000). Across both regimes, writing code without execution offers scaffolding proportional to the model's ability to absorb the cost of translation: on underpowered models the cost dominates and accuracy degrades; on capable models the cost is negligible and efficiency improves; in neither regime does syntax substitute for execution. H2 is strongly confirmed across every model tested. On accuracy: 56% $\rightarrow$ 100% on Sonnet 3.5, 60% $\rightarrow$ 100% on Gemini 2.5 Flash. On efficiency: 1.042 $\rightarrow$ 1.000 on Gemini 3 Flash. The extreme case is Sonnet 4.6, where CODE_mental never produces an artifact at all (substrate exhaustion) while CODE_executed reaches 100% accuracy. The closed verification loop is the only mechanism that achieves deterministic fidelity in every regime tested. To state this precisely: syntax is a *precondition* for Operability (it makes artifacts executable), but the value of Operability comes from the joint operation of execution, semantic persistence, and interpretable feedback in a closed loop, not from syntax itself. Structure without computation provides at best partial scaffolding, never deterministic fidelity.

RQ3: Tool vs. Mechanism, and the Conditions for Generalization. The first half of RQ3 asks whether the advantage observed in the executed conditions is attributable to the

use of an external tool or to the closed-loop mechanism specifically. The CODE_mental and ASP_mental conditions adjudicate this: in both, the model produces the same kind of artifact (Python algorithm; ASP encoding) as in the executed condition, yet performance collapses (56% and 13% respectively). The artifact-generation capacity of the model is held constant across mental and executed conditions; what varies is whether the artifact is closed back through external execution and the resulting feedback returned to the model. The performance gap (44 and 84 percentage points respectively) is therefore not attributable to the artifact-generation capacity itself but to the closed loop through which the artifact's behavioral consequences re-enter the model's context. This is consistent with the framing of Operability as a property of the model–domain relation rather than of any individual tool. Experiment 3 sharpens this further. Thinking-augmented models conduct extensive self-referential reasoning that resembles, at the surface, the iterative refinement Operability requires from an external loop. Yet the effect persists: on Gemini 3 Flash, the CoT verbalizes a BFS and executes a different process, producing efficiency loss (Box 1); on Sonnet 4.6, the model attempts faithful internal simulation and exhausts its inference budget without emitting an artifact (Box 2). Neither failure mode resembles successful internal verification. The mechanism that grounds generation is therefore not "external interaction in general," nor "extended deliberation," but specifically the closed loop with deterministic external semantics — a structural property that cannot be approximated internally regardless of inference budget. This is the strongest form of the mechanism claim the data support: neither external tools used as one-shot evaluators (the execute-and-check column of Table 2) nor extensive internal thinking constitutes Operability; only the externally closed iterative loop does. This observation also makes the framework empirically falsifiable rather than tautological. Operability admits multiple instantiations of the mechanism that realizes its three structural conditions (Figure 1); the substrate may be native, external, model-generated, or internal. A natural objection is that the open menu of instantiations renders the framework non-falsifiable: anything that works can be retrofitted as "the mechanism." Experiment 3 answers this objection directly. The instantiation in panel (d) of Figure 1 — an internal closed loop within the model's own inference — is a concrete empirical hypothesis: that thinking-augmented architectures realize the three conditions internally through extended deliberation. Box 1 and Box 2 falsify this hypothesis on the two models tested. Gemini 3 Flash declares one algorithm and reports the trace of another; Sonnet 4.6 commits to faithful internal simulation and exhausts its substrate without producing an artifact. Neither manifests the closure that the framework requires. The framework therefore predicts, and Experiment 3 confirms, that current thinking-augmented models do not realize the internal instantiation — a falsifiable claim about a specific configuration, not a definitional one about Operability in general. The second half of RQ3 asks under what conditions this advantage extends across different KR formalisms. Our two experiments illustrate two distinct modes of model–KR division of labor: in the maze, the LLM authors a solution that the interpreter executes (verification mode); in ASP, the LLM formalizes constraints that Clingo solves (computational delegation mode). Both modes satisfy the three structural conditions of Operability — executability, semantic persistence, interpretable feedback — and both yield the effect

at comparable magnitude. The two modes differ substantially in division of labor (who computes the solution: the model or the KR system) but converge in outcome. This is consistent with — though does not by itself prove — the framework's claim that the three structural conditions, rather than the division of labor, are the relevant generalization dimension. A direct test would require a domain where the structural conditions are jointly met but the division of labor differs further (e.g., a verifier that neither solves nor merely checks but proposes counterexamples). Table 8 maps the three conditions onto three further KR formalisms; whether the pattern holds for them remains an open empirical question.

	OWL/DL	**ASP**	**SHACL**
Execut.	Consistency checking	Stable model computation	Conformance checking
Persist.	Ontology ABox	Logic program	Knowledge graph
Feedb.	Justifications, MUS	Constraint violations	Validation reports

Table 8. Operability conditions mapped to KR tools.

Conjecture 1. *For any (model, mechanism, domain) configuration in which executability, semantic persistence, and interpretable feedback are jointly satisfied in a closed loop, and in which the model has the generative capacity to produce repairable artifacts compatible with the mechanism, Operability transforms Epistemic hallucination from a possible terminal failure into a traceable exploration. The Symbolic Bottleneck (Section 4.2) provides the operational converse: when the model's generative capacity is insufficient for the mechanism, the loop cannot close, and the effect does not emerge.*

Our experiments across two domains and four models provide initial evidence: the pattern holds for a programming interpreter and an ASP constraint solver — two systems with fundamentally different verification mechanisms — across two distinct modes of model–KR collaboration, and across model generations from non-thinking (Sonnet 3.5, Gemini 2.5 Flash) to thinking-augmented architectures (Gemini 3 Flash, Sonnet 4.6). We note one operational caveat: the Symbolic Bottleneck observed in the ASP experiment shows that the structural sufficiency stated in the conjecture presupposes that the model can produce repairable artifacts in the formalism at hand. This is a separate empirical question — the formalism's resource representation in training data — distinct from the structural properties of the verification loop itself. Validation across additional KR formalisms (in particular OWL/DL reasoners and SHACL validators) remains an active direction.

5.2 Limitations

Decidable ground truth. The maze and scheduling tasks have unambiguous ground truth by design: isolating the Operability effect requires domains where correctness

is decidable. Open-ended domains — where correctness is plural, contested, or itself negotiated — fall outside the scope of this work and remain a direction for future investigation.

Statistical power. On non-thinking models, our aggregate samples (270 maze trials across Sonnet 3.5 and Gemini 2.5 Flash; 90 ASP trials on Sonnet 3.5; 360 trials in total) support the main effect comparisons robustly: the CODE_executed vs. CODE_mental contrast is highly significant on both formalisms. The NL versus CODE_mental comparison reaches significance on neither individual cell ($p = 0.189$ on Sonnet 3.5 maze, $p = 0.506$ on ASP), although the directional pattern reproduces consistently across two model families on the maze task and within the ASP experiment, providing inferential support even where individual tests lack power. Experiment 3 probes how the Operability dynamics observed on non-thinking models manifest on thinking-augmented architectures, asking specifically whether extended thinking gives rise to an internal analogue of the external verification loop. With 60 trials, it is designed as a qualitative exploration of failure modes rather than as a statistical replication. This is a methodological choice rather than a resource constraint: the underlying question is mechanistic — not whether the outputs differ across conditions, which behavioural measurements can adjudicate, but *why* they differ — and the chain-of-thought, the only window onto thinking-model internal dynamics that frontier closed-weight architectures expose, is itself only a plausible account of those dynamics Turpin et al. (2023). A rigorous mechanistic study would require interpretability tools such as sparse autoencoders,[‡] which are not currently available for the frontier models that exhibit the relevant thinking capabilities. Quantitative replication of the behavioural findings (path efficiency, trace-fidelity failure rates, substrate-exhaustion incidence) across additional thinking architectures and larger samples is positioned as future work, as is interpretability-based mechanistic study should comparable open-weight thinking models become available. Cell-level samples (in particular, ASP Medium-difficulty with $n = 5$ trials per condition) are too small for safe stratification by difficulty; results should not be over-interpreted at the level of individual difficulty cells.

Case-study evidence in Experiment 3. Boxes 1 and 2 report single-trial close readings: the trace-fidelity failure on Gemini 3 Flash (seed 5003000) and the substrate-exhaustion failure on Sonnet 4.6. While the patterns they illustrate are consistent across the 10 trials per condition in Experiment 3, the boxes themselves are illustrative rather than statistically representative. Aggregate quantification of trace-fidelity failure rates — for example, the proportion of CODE_mental trials in which the reported path coincides exactly with the BFS output on the same written code — remains future work and is the natural next step for cross-model studies on thinking-augmented architectures.

Reasoning rather than factual hallucinations. Our experiments primarily test reasoning hallucinations — failures in state tracking and multi-step planning — which

[‡]We conducted exploratory SAE analyses on the Gemma 2B base model; these did not yield findings relevant to the present work.

represent the most challenging frontier for current state-of-the-art LLMs. Factual confabulation (inventing facts about the world), increasingly managed by frontier models through retrieval augmentation and post-training, was not directly tested. Our taxonomy suggests that Operability would help here too, through fact-checking verifiers and knowledge graph validators, but this remains an open question.

Cross-domain coverage. While our framework argues that Operability generalizes across KR tools, only one cross-domain experiment (ASP) was conducted. Direct experimentation with ontological reasoners (OWL/DL) and knowledge graph validators (SHACL) would further strengthen the generalization claim. Section 3.2 sketches how the three structural conditions map to these formalisms (Table 8); empirical validation across them remains future work.

The Symbolic Bottleneck. Operability assumes alignment between generator $\mathcal{G}$ and verifier $\mathcal{V}$ at two levels: syntactic validity and semantic repair capacity. While high-resource formalisms (Python) allow the feedback loop to sustain both, low-resource ones (ASP, SHACL) often fail in semantic repair specifically Ren et al. (2025). Our ASP experiment provides direct evidence: 47% of first-attempt encodings were valid; the model recovered to 97% over three iterations, but one trial illustrates the limit — despite valid ASP syntax and specific overlap-violation feedback, the model failed to generate the required constraint repair across all three attempts. This suggests Operability depends on the model's capacity to act on feedback, a capacity modulated by the formalism's representation in training data.

This dependency may look like a tension with the structural claim that Operability is a property of the model–mechanism–domain configuration: if generative capacity in the formalism modulates whether the configuration yields the predicted effect, the structural conditions appear to inherit a contingent dependency on training distribution. We address the tension directly. Conjecture 1 is structurally sufficient *conditional on* the model's generative capacity in the formalism, and this conditional clause is not a defect but the falsifiable empirical boundary of the framework, testable per formalism. A formalism for which a given model class produces sub-threshold first-attempt validity rates — operationally, well below the $\sim$80% Python rate we observe — predicts that Operability will fall short of deterministic fidelity even when the three structural conditions are formally satisfied. This is a prediction, not a hedge.

For the KR community, the implication is design-level: if Operability proves viable as a verification paradigm, the accessibility of a formalism to LLM-based generation becomes a first-class design criterion alongside expressiveness and tractability — operationalized, for example, by first-attempt validity rates that quantify generator–verifier alignment empirically.

Generalization of the trace-fidelity metric. Path efficiency as operationalized here is specific to tasks with a well-defined optimal solution length, such as the BFS-optimal path in the maze case. Extending the trace-fidelity diagnostic to domains without a unique optimal — open-ended planning, multi-objective optimization, or tasks admitting multiple correct solutions — requires a more general definition of fidelity between the model's reported solution and the deterministic output of its own written

artifact. The general schema (compare reported output against artifact-execution output) carries through, but calibration to task-specific notions of correctness remains an open methodological question.

6 Conclusion

The central contribution of this work is the characterization of the structural conditions under which a generative system can achieve epistemic grounding. Operability is not merely a static feature of a domain, but a property of the model–mechanism–domain configuration, which can be actively pursued. It is defined by three jointly necessary conditions—executability, semantic persistence, and interpretable feedback—that ensure the symbolic meaning of a generated artifact is carried forward in a closed loop. When these conditions are met, the iterative interaction between a probabilistic generator and a deterministic substrate transforms hallucination from an epistemic liability into a correctable exploration. Within this characterization, we posed three research questions and offer the following answers.

RQ1: The Taxonomy as Diagnostic Framework. Epistemic hallucinations appear architecturally distinct from Structural errors and Training-induced biases in their response to symbolic verification: across four models and two domains, external verification eliminates these failures while syntactic intervention without execution does not. The provenance failure underlying them persists across model generations, with surface manifestations that migrate from accuracy to efficiency to substrate exhaustion as model capability increases. Whether Structural and Training-induced hallucinations resist the same intervention — the complementary test of taxonomic selectivity — remains an open empirical question.

RQ2: Syntax alone is not sufficient. The combination with a semantically closed verification loop, not the syntactic form of the artifact alone, is the source of the Operability effect. In every regime tested, the executed condition reaches deterministic fidelity while the mental condition does not: the gap is robust to model lineage, formalism, and the presence or absence of extended thinking. Syntax is a precondition for executability, not a source of grounding.

RQ3: Tool vs. Mechanism, and the Conditions for Generalization. The effect is attributable to the closed-loop mechanism rather than to the use of an external tool as such. The mental conditions hold artifact generation capacity constant and remove only the loop, with collapse in performance. Our cross-model probe finds no evidence that extended thinking provides an internal analogue of this loop, suggesting that the relevant property is the closed external loop with deterministic semantics, not "extended deliberation" or external tool use in general. The structural conditions of Operability — executability, semantic persistence, interpretable feedback — govern the effect, rather than the specific division of labor between model and KR system.

If hallucination is a mathematical consequence of statistical inference for calibrated models Kalai and Vempala (2024) and in-context processing functions as implicit weight modification Dherin et al. (2025), the solution is not hallucination-free generation but environments that detect and correct hallucinations through closed-loop symbolic

verification. The contribution of Knowledge Representation in this framing is not as an alternative to probabilistic generation but as its complement: KR systems provide the deterministic verification through which probabilistic exploration becomes epistemically grounded. The Symbolic Bottleneck observed in our ASP experiment indicates that this complementary role is not free — for Operability to extend across formalisms, the formalism must be sufficiently manipulable by the generator, which depends on training-data representation, intermediate translation layers, or formalisms designed with LLM-accessibility in mind.

The broader implication: Epistemic hallucinations are not defects of a probabilistic system, but consequences of its architecture. They are not eliminated by making the system less probabilistic, but contained by giving it symbolic counterparts in which its explorations can be grounded — transforming hallucination from an epistemic liability into an iterative step toward verifiable output, where the structural conditions of the domain admit it. The qualifying clause — *where the structural conditions of the domain admit it* — is what defines Operability as a property of domains rather than of models, and it leads to two open questions.

Future directions. Two trajectories invite further investigation. The first is a mechanistic interpretability study of the Operability effect on thinking-augmented architectures. Specifically, directly examining internal representations via sparse autoencoders—once open-weight thinking models of comparable capability become available—would complement the behavioral and CoT-level evidence from Experiment 3. This would provide a mechanistic explanation for the epistemic appropriation documented behaviorally in this paper. The second, and most ambitious trajectory, concerns the extension of Operability beyond domains with a decidable ground truth. The framework as presented requires three structural conditions to be jointly met in a closed loop. While this configuration is native to domains such as coding, it does not naturally apply to open-ended generation. Can the properties empirically demonstrated in operable domains be transferred to open-ended domains through an intermediate generation step in an operable domain? Addressing this question requires a formal theoretical analysis of domain properties that extends beyond our current empirical findings, remaining an open challenge for future work.

Supplementary Materials

All code, data, prompts, and validation scripts are available at: https://github.com/AndoSan84/operability

References

Alansari A and Luqman H (2025) Large language models hallucination: A comprehensive survey. *arXiv preprint arXiv:2510.06265* .

Cheng S, Pan L, Yin X, Wang X and Wang WY (2024) Understanding the interplay between parametric and contextual knowledge for large language models. arXiv:2410.08414.

Cossio M (2025) A comprehensive taxonomy of hallucinations in large language models. arXiv preprint 2508.01781.

d'Avila Garcez A and Lamb LC (2023) Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review* 56(11): 12387–12406. DOI:10.1007/s10462-023-10448-w.

De Raedt L, Dumančić S, Manhaeve R and Marra G (2020) From statistical relational to neural-symbolic artificial intelligence. In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 4943–4950.

DeVilling B (2025) The mirror loop: Recursive non-convergence in generative reasoning systems.

Dherin B, Munn M, Mazzawi H, Wunder M and Gonzalvo J (2025) Learning without Training: The Implicit Dynamics of In-Context Learning. arXiv preprint 2507.16003.

Gao L, Madaan A, Zhou S, Alon U, Liu P, Yang Y, Callan J and Neubig G (2023) PAL: Program-aided language models. In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Gurnee W and Tegmark M (2024) Language models represent space and time. In: *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Hitzler P, Eberhart A, Ebrahimi M, Sarker MK and Zhou L (2022) Neuro-symbolic approaches in artificial intelligence. *National Science Review* 9(6): nwac035. DOI:10.1093/nsr/nwac035.

Huang J, Chen X, Mishra S, Zheng HS, Yu AW, Song X and Zhou D (2024) Large language models cannot self-correct reasoning yet. In: *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Huang L, Yu W, Ma W, Zhong W, Feng Z, Wang H, Chen Q, Peng W, Feng X, Qin B and Liu T (2025) A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43(2): 1–55. DOI: 10.1145/3703155.

Ji Z, Lee N, Frieske R, Yu T, Su D, Xu Y, Ishii E, Bang Y, Chen D, Dai W, Chan HS, Madotto A and Fung P (2023) Survey of hallucination in natural language generation. *ACM Computing Surveys* 55(12): 1–38.

Kadavath S et al. (2022) Language models (mostly) know what they know. arXiv:2207.05221.

Kalai AT, Nachum O, Vempala SS and Zhang E (2025) Why language models hallucinate. *arXiv preprint arXiv:2509.04664* .

Kalai AT and Vempala SS (2024) Calibrated language models must hallucinate. In: *Proc. 56th Annual ACM Symposium on Theory of Computing (STOC)*. pp. 160–171. DOI:https://doi.org/10.1145/3618260.3649777.

Kang B, Yue Y, Lu R, Lin Z, Zhao Y, Wang K, Huang G and Feng J (2024) How far is video generation from world model: A physical law perspective. arXiv preprint arXiv:2411.02385.

Kautz HA (2022) The third AI summer: AAAI Robert S. Engelmore memorial lecture. *AI Magazine* 43(1): 105–125. DOI:10.1002/aaai.12036.

Khalifa M, Wadden D, Strubell E, Lee H, Wang L, Beltagy I and Peng H (2024) Source-aware training enables knowledge attribution in language models. In: *Conference on Language Modeling (COLM)*.

Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, tau Yih W, Rocktäschel T, Riedel S and Kiela D (2020) Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems* 33: 9459–9474.

Li K, Hopkins AK, Bau D, Viégas F, Pfister H and Wattenberg M (2023) Emergent world representations: Exploring a sequence model trained on a synthetic task. In: *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Madsen A, Chandar S and Reddy S (2024) Are self-explanations from large language models faithful? In: *Findings of the Association for Computational Linguistics: ACL 2024*.

Marcus G (2020) The next decade in AI: Four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177* .

Min S, Krishna K, Lyu X, Lewis M, tau Yih W, Koh PW, Iyyer M, Zettlemoyer L and Hajishirzi H (2023) FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In: *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Rawte V, Sheth A and Das A (2023) A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922* .

Ren L, Xiao G, Qi G, Geng Y and Xue H (2025) Can LLMs solve ASP problems? Insights from a benchmarking study. In: *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*. pp. 653–658.

Schick T, Dwivedi-Yu J, Dessì R, Raileanu R, Lomeli M, Hambro E, Zettlemoyer L, Cancedda N and Scialom T (2023) Toolformer: Language models can teach themselves to use tools. In: *Advances in Neural Information Processing Systems 36 (NeurIPS)*.

Sharma M et al. (2024) Towards understanding sycophancy in language models. In: *Proc. 12th International Conference on Learning Representations (ICLR)*.

Shin A and Kaneko K (2024) Large language models lack understanding of character composition of words. In: *ICML 2024 Workshop on LLMs and Cognition*.

Stechly K, Valmeekam K and Kambhampati S (2025) On the self-verification limitations of large language models on reasoning and planning tasks. In: *International Conference on Learning Representations (ICLR)*.

Tighidet Z, Mogini A, Mei J, Piwowarski B and Gallinari P (2024) Probing language models on their knowledge source. In: *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*.

Tonmoy SMTI, Zaman SMM, Jain V, Rani A, Rawte V, Chadha A and Das A (2024) A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313* .

Turpin M, Michael J, Perez E and Bowman SR (2023) Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In: *Advances in Neural Information Processing Systems*.

Vasileiou SL and Yeoh W (2025) TRACE-CS: A hybrid logic–LLM system for explainable course scheduling. In: *Proc. 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Vu T, Iyyer M, Wang X, Constant N, Wei J, Wei J, Tar C, Sung YH, Zhou D, Le Q and Luong T (2023) FreshLLMs: Refreshing large language models with search engine augmentation. *arXiv preprint arXiv:2310.03214* .

Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K and Cao Y (2023) ReAct: Synergizing reasoning and acting in language models. In: *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Zhang M, Press O, Merrill W, Liu A and Smith NA (2024) How language model hallucinations can snowball. In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*.