Neurosymbolic Artificial Intelligence
XX(X):2–35
©The Author(s) 0000
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Operability: On the Conditions for Epistemic Appropriation in LLMs

Andrea Gasparro[1] and Paolo Ciancarini[2]

Abstract

Systems that improve iteratively on their own output rest on a precondition: the ability to build on what they produce. Large language models (LLMs) make this precondition non-trivial. On tasks that exceed what a single inference can compute, an LLM still produces a complete answer, including demonstrably impossible ones. Since LLMs do not reliably introspect their own reasoning (Turpin et al. 2023), we focus on the structural properties of the configuration in which the model operates. We characterize a set of conditions under which such an answer can be repaired locally, each repair building on the last: the model produces an artifact whose meaning is fixed by a deterministic executor, which propagates the consequences of a local change (executability, semantic persistence), and the executor reports what failed (interpretable feedback). We propose that when these three conditions hold—a property we term Operability—the model encounters the behavioral consequences of its own generation and can appropriate its output, building on it instead of producing something new. In pre-registered experiments on job-shop scheduling with an Answer Set Programming solver, with the repair loop held fixed and in the regime where single-shot solutions fail, diagnostic feedback repairs more failed encodings than pass/fail feedback at an equal number of attempts (60% vs. 27%). Under the same loop, an executable encoding is solved more often than a prose solution (78% vs. 20%). An exploratory analysis shows how repair proceeds in the two media: the model edits its encoding locally, keeping two thirds of its rules between attempts (against 17% shared by independent encodings), and the solver propagates the consequences; in prose the model also repairs where it is told, but must propagate the consequences itself, and each repair introduces new violations. On maze-solving, executing the model's code succeeds where prose and mentally traced code fail, on three models. Exploratory probes of thinking-augmented models show reasoning traces diverging from the code the model itself has written. We situate these conditions within a causal taxonomy of hallucinations and discuss implications for knowledge representation systems.

Keywords
Neuro-Symbolic AI, Hallucination, Verifier-in-the-loop, Answer Set Programming, Knowledge Representation, Operability

1 Introduction

1.1 Hallucination and Bounded Computation

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. Recent work has begun to ask a different question: under which conditions an erroneous output can be detected and corrected, rather than only made less likely (Huang et al. 2024; Stechly et al. 2025; Kambhampati et al. 2024).

A complementary view, which we adopt here, concerns computation rather than knowledge. Tasks such as path-finding with state or constrained scheduling require a search whose size grows quickly with the instance; within a bounded inference budget, a model can describe a correct procedure and still fail to carry it out (Ren et al. 2025; Stechly et al. 2025). In our experiments, the answer the model produces in that case is nonetheless complete: the model does not decline to answer.

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 work that treats hallucination as a property to be managed rather than only reduced (Huang et al. 2025; Tonmoy et al. 2024). The question we ask is: *under what conditions can a model repair its own erroneous answer, so that each repair builds on the last?*

1.2 The Gap

Systems that pair an LLM with a sound external verifier are now common: the model generates, an executor or a critic checks, and the report is fed back (Kambhampati et al. 2024; Gao et al. 2023; Wang et al. 2024). This work shows *how* to build such loops. It does not say *when* a loop lets the model build on its own previous output rather than produce a new one, nor which properties of the loop make the difference. Experiments with such loops typically vary the loop itself—critics, prompts, number of rounds; to

[1]Università di Bologna, Italy
[2]Università di Bologna, Italy

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

answer this question, the loop must be held fixed while what enters it is varied, one factor at a time. This is what we do.

Most existing work focuses on *detecting* or *reducing* hallucinations through prompting techniques, retrieval augmentation, or improved training data (Huang et al. 2025; Tonmoy et al. 2024), and 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. To situate the problem we use a causal taxonomy, which distinguishes hallucinations by their abstract cause: *Structural* hallucinations arise from the computational process itself (tokenization, errors propagating along a chain of reasoning); *Epistemic* hallucinations arise from how the model manages knowledge and its provenance (confabulation, unfaithfulness to the context); *Training-induced* hallucinations are behaviors incentivized by post-training (sycophancy, completion bias). We use it as a conceptual frame (Section 3); our experiments do not test it.

1.3 Thesis

Our starting point is an observation, which the experiments of this paper examine in two task domains: models appear to reason more reliably when the artifacts they generate during a process retain their meaning across its steps, and to 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 does not, in our experiments, ground the answer. 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) argues 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 conditioned by the specific properties of the artifact. This is not a property a formalism possesses intrinsically: natural language affords it weakly—in our experiments, the model repairs a prose schedule where it is told, but must propagate the consequences itself, and new violations accumulate—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. In the terms of Definition 1, this is semantic persistence; what it is meant to enable—corrections that build on the previous state rather than starting from scratch—we call appropriation, and this paper reports an exploratory measure of it (Section 4.2).

The question has a broader motivation. A system that improves iteratively on its own output—the basis of autonomous self-improvement—must first be able to take up what it has produced and build on it rather than start again. This paper studies that prerequisite in its most basic form, over a few attempts at repairing a single answer (Section 6 returns to longer horizons).

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 determine whether grounding occurs, and we treat the answer as a property to be characterized rather than a technique to be optimized.

When a model's answer is wrong, what matters next is whether it can be repaired. We call *Operability* the property that makes this possible: a structural property of the model–mechanism–domain configuration, present when three conditions are jointly realized in a closed loop (Definition 1)—*executability*, a deterministic procedure evaluates what the model produced; *semantic persistence*, the artifact keeps its meaning across attempts, and the executor, not the model, propagates the consequences of a change; and *interpretable feedback*, the evaluation reports what failed, in a form the model can act on. In its paradigmatic form this loop is execution: when the model's output is run by a deterministic oracle, its *behavior* becomes a ground truth the model does not compute itself. Critically, this is not an external correction bolted onto an otherwise unchanged model. Dherin et al. (2025) show that context entering a transformer acts as a transient, low-rank modification of the model's effective 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 conditions the computation that produces the following artifact. 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: what it returns becomes part of the context from which the next output is generated. This is what we call *epistemic appropriation*[*]: the model does not merely receive a verdict on its output; it continues from a context in which the consequences of

[*]We use 'appropriation' for a precise, observable property: across successive attempts, the model modifies the artifact it has already produced instead of producing a new one, so that earlier repairs persist (measured, in an exploratory analysis, in Section 4.2). The term echoes the pragmatist sense of taking-up-and-making-one's-own (cf. Brandom 1994, Haugeland 1998) without committing to its philosophical machinery.

that output condition its next attempt, and it can modify what it has produced instead of producing something new.

Where Operability is present, we expect erroneous answers to become repairable candidates, tested and refined against a concrete ground truth; in our experiments, they largely did. Where it is absent—for instance when what persists across attempts does not preserve the meaning of the output—errors are replaced rather than repaired: in our experiments, a prose solution loses its errors from one attempt to the next only to acquire new ones.

The verification component in this framework functionally plays the same role as a Knowledge Representation system. In our experiments, the Python interpreter and the Clingo solver serve this role: they fix the semantics of the artifact, perform deterministic symbolic computation, and return interpretable feedback. But they are instances 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 can provide the same three properties: executability, semantic persistence, and interpretable feedback. This positions KR not as an alternative to probabilistic generation, but as its 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 offers a complement that does not depend on this narrative. 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, in Operability the closed-loop structure itself provides the explanation of why a candidate was rejected or accepted. 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 (Interpretable feedback): At an equal number of attempts, does diagnostic feedback repair more failed artifacts than a pass/fail verdict, and how do both compare with single-shot execution? (Experiment 2A)

RQ2 (Syntax vs. execution): Does formal syntax without execution improve over natural language, or does the advantage require execution? (Experiment 1)

RQ3 (Medium under an identical loop): Under the same repair loop, does the medium in which the solution is held—prose or an executable encoding—change how often it is solved and repaired, and does the repair build on the previous attempt? (Experiment 2B) Experiment 3 probes, qualitatively, whether extended thinking acts as an internal executor of the model's own code.

1.5 Contributions

This paper makes three contributions.

(1) The Operability framework, defined by three structural conditions—executability, semantic persistence, and interpretable feedback—under which a generative model can repair its own output locally, each repair building on the last, through iterative interaction with a substrate that realizes those conditions. Each condition comes with an operational criterion, checked on the configuration, and its effect with a measure on the model's behavior (Section 3). In the configurations we examine, the substrate is an external symbolic executor; whether future architectures might realize the same conditions through internal mechanisms remains an open empirical question, explored qualitatively in Experiment 3 (Section 4.3). Operability is framed as a property of the model–mechanism–domain configuration rather than of any individual component or tool: tool-augmented systems share some of the same functional ingredients (external process, returned output), but the framework proposes that it is the configurations jointly satisfying the three conditions in a closed loop that let the model build on its own previous output; Experiments 2A and 2B test this for the feedback and the medium.

(2) Pre-registered experimental evidence in the regime where single-shot solutions fail. On job-shop scheduling with the Clingo ASP solver, diagnostic feedback repairs more failed encodings than pass/fail feedback at an equal number of attempts (60% vs. 27%), and under an identical repair loop an executable encoding is solved far more often than a prose solution (78% vs. 20%). On maze-solving, executing the model's code succeeds where prose and mentally traced code fail, on three models from two families.

(3) A causal taxonomy of hallucinations, classifying failures by their abstract cause—architecture, approximation and extrapolation, reward design—as a conceptual frame for choosing interventions. We do not claim that the categories are separable from the failures observed on the tasks we test.

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 evaluation, whether by executing a procedure the model wrote or by solving the constraints the model formalized.

The remainder of this paper is organized as follows. Section 2 reviews related work on hallucination taxonomies, the opacity of LLM inference, neuro-symbolic approaches, and systems that pair an LLM with an external verifier. Section 3 presents our causal taxonomy, the Operability framework, and the 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	**Causal 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 *causal origin* (why it occurs), yielding conceptually 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 point of intervention—and specifically highlighting Epistemic hallucinations as the category for which external verification is the natural intervention.

Our taxonomy is not merely descriptive but *causal*: it names abstract causes, not the circuits or representations that implement them, 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, current models have no reliable internal means of distinguishing the two at output time (Tighidet et al. 2024; Cheng et al. 2024), which makes external verification the natural intervention.

The taxonomy serves here as a conceptual frame. Our experiments do not test it: on the tasks we use, the failures we observe could be described both as Structural (constraint forgetting in a long chain) and as Epistemic (unverified intermediate states), and we do not claim to separate them. What the experiments test is a complementary question—what makes an erroneous answer repairable—which does not depend on the category of the error. When a chain of computation can be externalized into an executable artifact, failures that the taxonomy would classify as Structural—such as constraint forgetting—also respond to execution, as in our experiments: the categories guide the choice of intervention; they do not exclude others.

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

A body of empirical evidence indicates that current LLMs do not 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) show that, in their reasoning and planning domains, self-critique degrades performance while a sound external verifier improves it; they report that re-prompting with a sound verifier—the same prompt resampled, without the history of previous attempts—retains most of the benefit of richer critique overall, while in their planning domain (Blocksworld) first-error feedback outperformed a binary verdict given together with the previous attempts (87% vs. 60%, with resampling at 68%); we test the comparison between the two kinds of feedback on our task (RQ1). 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. Models may also return to earlier erroneous decisions even after correct information enters the context (Lu et al. 2025); we measure this directly, as the rate at which repaired violations reappear (Section 4.2). Madsen et al. (2024) find that self-explanations are convincing but often wrong, with faithfulness depending on explanation type, model, and task.

A related limitation concerns provenance. 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.

Taken together, these findings suggest that even where the distinction between verified and interpolated content is represented, it is not reliably available to the model's own output decision; this motivates externalizing verification.

A theoretical result connects these observations to the loop we study. Dherin et al. (2025) show that in-context learning acts as a low-rank modification of the weights of

the first MLP layer: in-context processing is a transient form of adaptation, in which each forward pass modifies the model's effective behavior based on context. This makes the content that re-enters the context consequential for subsequent generation, which is the lever Operability relies on.

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. Closing a loop between a neural learner and symbolic knowledge, in which each refines the other, goes back at least to the neurosymbolic learning cycle of Towell and Shavlik (1993). 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.

Within an iteration, the configurations we study belong to the Neural | Symbolic class of Kautz's taxonomy: a neural system produces an output that a symbolic system then reasons about (Kautz 2022); across iterations, the closure of the loop follows the neurosymbolic learning cycle. We do not propose a new 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. Within this class, we characterize the *structural conditions of the coupling* under which the loop lets the model build on its own previous output rather than produce a new one. The same wiring can behave very differently depending on these conditions—in our experiments, depending on the medium in which the model's output is held (Section 4.2). The mechanism achieving the conditions may be native to the domain, externally provided, or generated at runtime by the model itself (Section 3).

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 are consistent with the pattern our framework describes: LLMs generate plausible candidates, and symbolic KR systems verify them. Unlike specific hybrid systems such as TRACE-CS, our aim is not a system but the conditions under which such systems work. The implication is not that any external tool grounds generation, but, we propose, that grounding depends on whether 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.

2.3 LLMs with External Verifiers in the Loop

Several systems close a loop between an LLM and a sound external checker. LLM-Modulo (Kambhampati et al. 2024) pairs a generator with external critics in a generate–test–critique loop whose back-prompts pool the critics' reports; it has been instantiated on planning and scheduling (Gundawar et al. 2024). Counterexample-guided inductive synthesis with LLMs and SMT solvers (Jha et al. 2023), PEIRCE's LLM-driven neuro-symbolic refinement (Quan et al. 2025) and agents that act through executable code (Wang et al. 2024) follow the same pattern; Yang et al. (2024) survey code as a medium for LLM reasoning. Program-aided reasoning (Gao et al. 2023) is the single-shot case. These systems show how to build the loop and that it helps. Our question is a different one: which conditions of the loop determine whether the model builds on its own previous output. To answer it, our experiments hold the loop fixed and vary one factor at a time: the content of the feedback, then the medium. The levels we compare correspond to published configurations, on identical instances: single-shot execution (program-aided reasoning), a pass/fail verdict with the previous attempt in context, as in the binary-feedback condition of Stechly et al. (2025), and a critique loop that names the violated constraints, as in LLM-Modulo (Experiment 2A); and the same critique loop with the solution held in two different media (Experiment 2B).

3 Method and Framework

3.1 A Causal Taxonomy of Hallucinations

We propose a taxonomy that classifies hallucinations by their abstract cause, 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 conceptual tool oriented toward intervention, identifying the cause of a hallucination in order to prescribe the appropriate class of remedy. The categories are not mutually exclusive descriptions of an observed failure: the same failure can admit more than one causal reading, and we do not use the taxonomy to classify the failures observed in our experiments.

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.

Epistemic hallucinations are tied to how knowledge is represented: the same parameters encode verified facts and plausible interpolations, and the distinction is not reliably available to the model's output decision (Tighidet et al. 2024). External verification is therefore the natural intervention for this category.

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.*

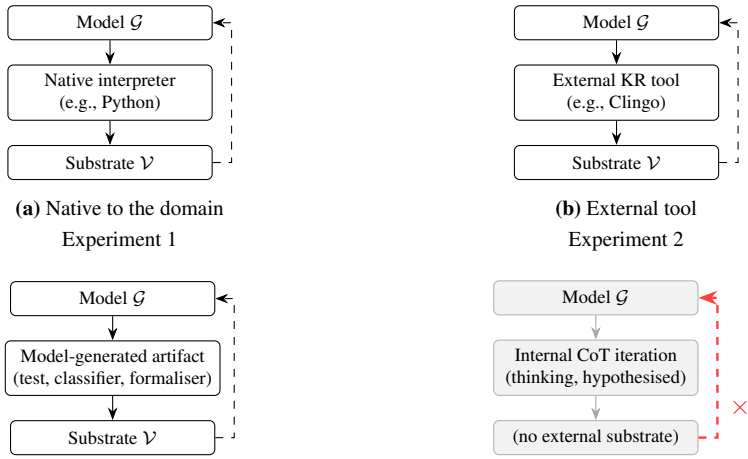
The mechanism of clause (ii) can have different origins, illustrated in Figure 1. A model–mechanism–domain configuration exhibits *Operability* when three conditions are jointly met:

1. **Executability**: a decision procedure deterministically evaluates the artifact and establishes whether it satisfies the task's requirements.

2. **Semantic Persistence**: the artifact preserves its meaning across iterations: it re-enters the context, its meaning is fixed by the executor, and the executor—not the model—propagates the consequences of a local change. This is what makes cumulative refinement possible, each correction building on the previous state rather than starting from scratch.
3. **Interpretable Feedback**: the evaluation's output is structured in a form that the model can process and act upon for correction—error messages, test results, constraint violations.

Operational criteria. Each condition is checked a priori, on the configuration; its effect is measured on the model's behavior. *Executability* holds when a decision procedure evaluates the artifact deterministically (here: the Python interpreter; Clingo together with a specification checker). *Interpretable feedback* holds when the evaluation names what failed—the violated constraint, the failing step—rather than only whether the artifact failed; its effect is tested through a controlled contrast with pass/fail feedback at an equal number of attempts (Experiment 2A). *Semantic persistence* holds when the previous artifact re-enters the context, its meaning is fixed by the executor, and the executor propagates the consequences of a local change: a Clingo encoding satisfies it; a schedule written in prose does not, since the model must propagate any change it makes. What persistence is meant to enable—appropriation—is measured on the model's behavior, as continuity between successive attempts: the share of the artifact kept verbatim, the share of reported violations removed per repair step, the number of new violations introduced, and the share of repaired violations that later return (Experiment 2B). These criteria are not a closed definition; they make the conditions checkable and their effects measurable in the domains we study.

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 $\mathcal{F}_i$ conditions the next generation. It does not constrain it, and nothing in the loop guarantees convergence; whether and how feedback is taken up is an empirical question (Section 4.2). In the experiments of this paper, C_i contains the previous artifact and its feedback only (last-step persistence), not the full history. Dherin et al. (2025) show that context entering a transformer acts as a transient, low-rank modification of the model's effective weights. Feedback that re-enters the context therefore conditions the next generation. This explains why the content of what re-enters matters; it does not by itself guarantee that the feedback is used, nor that it is used well. Two levels can be distinguished. Within a single forward pass, context acts as an implicit modification of the model's effective weights (Dherin et al. 2025). Across iterations, an agentic loop operates: $\mathcal{G}$ generates α_i, $\mathcal{V}$ evaluates it, and $\mathcal{F}_i$ enters the context. Operability concerns the second level: it names the conditions under which what re-enters is the deterministic



(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
not observed in Experiment 3

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) does not observe in the two thinking-augmented models probed. The framework is silent on which form realises the conditions; what matters is the conjunction.

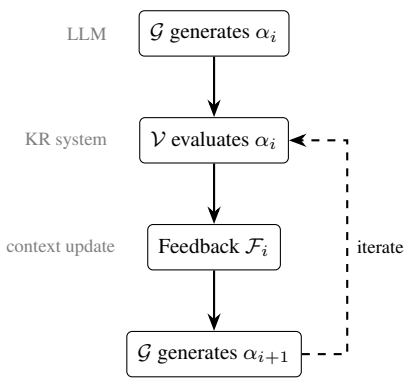


Figure 2. The Operability loop. $\mathcal{G}$ produces artifact α_i; the verifier $\mathcal{V}$ evaluates it and returns feedback $\mathcal{F}_i$, which enters the context for the next generation cycle.

	Tool-use	**Execute-and-check**	**Iterated, pass/fail**	**Operability**
LLM output verified?	No	Once	Yes	Yes
Feedback type	Informative (new data)	Result (pass/fail)	Verdict only	Diagnostic (what failed)
Iteration on artifact?	No	No	Yes	Yes
Semantic persistence?	N/A	N/A	Possible	Required
Epistemic relation	Augmentation	Evaluation	Repeated evaluation	*Appropriation*
Example	RAG, Toolformer	PAL; first attempt in Exp. 2A	Binary feedback (Stechly et al. 2025); pass/fail arm in Exp. 2A	LLM-Modulo; diagnostic arm in Exp. 2A

Table 2. Four levels of external interaction in tool-augmented LLM systems. All 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; iterated pass/fail feedback evaluates repeatedly without saying what failed; Operability identifies the structural conditions under which iterative external interaction yields *epistemic appropriation* rather than mere augmentation or evaluation. Experiment 2A compares the last three levels on identical instances.

consequence of the model's own artifact, so that the outcome of one attempt conditions the next. Experiment 2 measures how that content is taken up.

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 has not made explicit is which configurations of external interaction let the model build on its own output, and which yield only informational augmentation. Operability is our proposal for that account. The framework identifies, within tool-augmented systems, a structural condition: when an external process satisfies executability, semantic persistence, and interpretable feedback in a closed loop, the resulting interaction is what we call epistemic appropriation, rather than mere tool-use. We distinguish four levels of external interaction (Table 2); only the last 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; whether that verdict can be acted upon is what iteration adds.

Iterated pass/fail feedback, as in the binary-feedback condition of Stechly et al. (2025), adds iteration: the model receives its previous artifact and a verdict, and tries again. What it does not receive is what failed.

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 but lacks the iterative loop; iterated pass/fail feedback iterates on the artifact, but its feedback says only that the artifact failed, not what failed. Only Operability has all three jointly. Experiment 2A tests whether the content of the feedback makes a difference at an equal number of attempts.

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 not reliably accessible 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 is tested in Experiment 1.

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 that 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.

In the first two runs, 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, in a computation whose size grows quickly with the number of keys.

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; the path it prints is validated. 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 execution matters, CODE_executed should outperform CODE_mental.

The comparison was run three times, on three models. **Claude Sonnet**: 45 mazes, 15 per difficulty level—Easy (8×8 grids, 1 key, state space $\sim$128), Medium (10×10 grids, 2 keys, $\sim$400), and Hard (12×12 grids, 3 keys, $\sim$1152)—with up to three attempts per condition and pass/fail feedback after a failed attempt (each retry was a new call that included the original prompt and a pass/fail message, but not the previous answer), through the Claude command-line interface with the model alias `sonnet` and the interface's default settings (January 2026; neither the resolved model version nor whether extended thinking was active was recorded). **Gemini 2.5 Flash**, a model from a different lineage: 20 mazes (7 Easy, 7 Medium, 6 Hard), same protocol, through the Gemini command-line interface in non-interactive mode, with the model's default dynamic thinking (the session logs record reasoning tokens). **Claude Haiku 4.5** (`claude-haiku-4-5-20251001`, extended thinking disabled): 20 mazes of

12×12 cells with 4 keys from a generator with random wall, key and door placement, which prevents solving the maze by reasoning about its zones; one attempt per condition, paired on instances, through a hardened harness—no tools, an empty working directory for each call, and the model identifier checked on every call. The first two runs used agentic command-line interfaces, which affected them in ways we report with their results (Section 4.1); the third re-runs the comparison under the hardened protocol. 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: Feedback and Medium in a Repair Loop

To test whether Operability extends beyond programming to a genuine KR formalism, and to test two of its conditions separately, Experiment 2 uses job-shop scheduling with the Clingo ASP solver. 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). An encoding is evaluated by Clingo and by an independent specification checker, which reports each violated constraint. Greedy strategies fail when precedence chains force non-obvious orderings—paralleling the key-before-door constraint in the maze task.

Both parts of the experiment were pre-registered: design, prompts, decision rules and analysis script were committed before the calls they govern, and every call is recorded with its raw response (Supplementary Materials). They were run on Claude Haiku 4.5 (`claude-haiku-4-5-20251001`) with extended thinking disabled, through the hardened harness described for Experiment 1. Every retry carries the previously evaluated artifact and its feedback (last-step persistence).

Choice of model and regime. The task of the original design was kept. On a current Sonnet model (Claude Sonnet 5) the task is at ceiling: 43 of 46 first-attempt encodings were valid, and where the first attempt succeeds there is nothing to repair. Instance size does not change this—the failure rate of the encoding did not grow over five increasing instance sizes—since the same program serves every size. With extended thinking, Haiku 4.5 is also at ceiling (13 of 13). The experiment therefore uses Haiku 4.5 without extended thinking, a model that its developer reports as offering coding performance similar to Claude Sonnet 4 (Anthropic 2025), in the regime in which single-shot solutions fail and repair can be studied. These choices were recorded in the pre-registration before any outcome comparing the arms was computed.

Experiment 2A: interpretable feedback. Each instance receives one shared first attempt. Instances whose encoding runs but yields an invalid schedule, or no answer set, branch into two arms with two further attempts each: a *pass/fail* arm, told only that the schedule is not valid, and a *diagnostic* arm, told which constraints are violated. The two arms differ only in the feedback block, which is checked byte for byte. The primary outcome is the number of attempts to success (one-sided Wilcoxon signed-rank test, with a pre-registered threshold of half an attempt on the Hodges–Lehmann estimate);

the secondary outcome is success within three attempts (McNemar). The target was 30 branched instances (18 jobs, 5 machines, 18 precedences).

Experiment 2B: the medium under an identical loop. 200 fresh instances (16 jobs, 4 machines, 16 precedences) are each solved in two media: a schedule written in prose (NL), and a Clingo encoding executed by the solver (ASP). The loop is the same in both: up to three attempts, diagnostic feedback after any failure, and the previously evaluated object in the prompt. The primary outcome is success within three attempts (exact McNemar test, with a pre-registered threshold of 15 percentage points); secondary outcomes are first-attempt success and repair among the instances that failed the first attempt in both media. Experiment 2B uses 16 jobs rather than the 18 of Experiment 2A because of a pre-registered floor rule: at 18 jobs the prose condition solved 0 of 12 pilot instances, and the run moved once to the next easier point of the calibration grid. Two departures from the pre-registration are recorded in the repository: Experiment 2B and the Haiku run of Experiment 1 were decided after Experiment 2A had been analysed, and one interim look at Experiment 2A, at 21 of 30 branched instances, was taken before its target was reached; neither changed a design parameter or a decision rule.

3.6 Experiment 3: An Exploratory Probe of Thinking-Augmented Models

Experiment 2 and the Haiku run of Experiment 1 use a model with extended thinking disabled, and are 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, this would indicate that, in the models tested, intra-model reasoning does not substitute for externally closed verification. If instead the effect disappears, Operability would emerge as a property that sufficient inference budget can internalize. Experiment 3 probes this question qualitatively, on two thinking-augmented models: Gemini 3 Flash Preview, run through the Gemini command-line interface in non-interactive mode, in which the shell tool is not available to the model (the session logs record no tool call in the NL and CODE_mental conditions); and Claude Sonnet 4.6, run through the Claude command-line interface with its default configuration.

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 has the length of the shortest path. 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). With accuracy saturating, the evidence is descriptive: aggregate efficiency and close reading of individual reasoning traces (Section 4.3).

4 Results

4.1 Experiment 1: Syntax vs. Execution on the Maze

Condition	**Easy** (8×8, 1 key)	**Medium** (10×10, 2 keys)	**Hard** (12×12, 3 keys)	**Overall**	**No model answer**
NL	13/15	14/15	5/15	32/45	0
CODE_mental	15/15	6/15	4/15	25/45	2
CODE_executed	15/15	15/15	15/15	45/45	0

Table 3. Maze accuracy on Claude Sonnet (Claude command-line interface, model alias `sonnet`, January 2026; 15 mazes per difficulty level, up to three attempts per condition with pass/fail feedback). In the CODE_mental condition, 27 of 103 attempts ended in an API error caused by the interface's tool calls; in two trials no attempt returned a model answer. These trials are counted as failures, as in the original analysis, and reported in the last column.

Condition	**Valid paths**	**Failures**
NL	0/20 (0–16%)	16 wall traversals, 2 non-adjacent steps, 2 doors without key
CODE_mental	0/20 (0–16%)	15 wall traversals, 3 non-adjacent steps, 2 doors without key
CODE_executed	19/20 (76–99%)	1 program crash

Table 4. Maze accuracy on Claude Haiku 4.5 without extended thinking, hardened protocol (20 mazes, 12×12 with 4 keys, one attempt per condition, paired on instances; 95% Wilson intervals in parentheses). Every valid executed path has optimal length.

Execution vs. no execution. On Claude Sonnet (Table 3), CODE_executed solves all 45 mazes—39 of them at the first attempt that returned a model answer—against 32 of 45 for NL and 25 of 45 for CODE_mental (exact McNemar test on the paired mazes, CODE_executed vs. CODE_mental, $p < 0.0001$). H2 is supported. The difference between NL and CODE_mental is not significant (exact McNemar test, $p = 0.21$, excluding the two trials without a model answer), and we do not claim it: H1 receives

no support in either direction. Without execution, success falls as the state space grows: CODE_mental solves 15 of 15 Easy mazes, 6 of 15 Medium and 4 of 15 Hard; NL holds on Easy and Medium (13 and 14 of 15) and falls on Hard (5 of 15). The executed condition is unaffected by size.

On Claude Haiku 4.5, under the hardened protocol and on harder mazes (Table 4), CODE_executed solves 19 of 20 mazes at the first attempt, every one along an optimal path, while both non-executed conditions are at floor (0 of 20; McNemar, CODE_executed vs. CODE_mental, $p < 0.0001$). At floor, the comparison between prose and mentally traced code cannot be measured. On the maze, the advantage of execution is obtained at the first attempt: it is the delegation of the search to the interpreter, not repair through a loop.

The non-executed answers are complete, and none declines to answer, yet they are often impossible: on Haiku, 23 of the 40 non-executed paths are shorter than the shortest valid path, and most fail within the first few steps. We note that the prompts required a path in a fixed format and did not offer an option to abstain (Section 5.3).

Correct code, different path. Both Claude runs show a dissociation. Executed after the run, the programs the model wrote in the CODE_mental condition are correct in 65 of 65 attempts on Sonnet and in 19 of 20 on Haiku 4.5: they return a valid path.[†] Yet the path the model reports coincides with the output of its own program in only 2 of 65 attempts on Sonnet, and in none on Haiku. Whatever these models have learned about breadth-first search, in these runs they do not apply it, within a single inference, to the specific input.[‡] The interpreter provides exactly what the model does not: not a representation of the algorithm, but its execution on this input.

Writing the procedure and carrying it out are therefore distinct capacities: in these runs, the models do the first and, within a single inference, not the second. This is consistent with a bounded-computation account of the failures (Section 5.2), and it is the situation in which the question of repair arises: an answer has been produced, it is wrong, and the model has not declined to give it.

Gemini 2.5 Flash. On Gemini 2.5 Flash, a model from a different lineage (Table 5), the session logs of the command-line interface show that in most NL and CODE_mental sessions the model tried to use the interface's tools—the shell was not available—and that in 22 sessions it delegated part of the work to a sub-agent of the interface. Code was actually executed once, in one NL trial, which we do not count as an NL success. In five CODE_mental trials and one NL trial the model returned no path, stating that it could

[†]This check was not pre-registered. The code of each CODE_mental response was extracted and executed in isolation on its maze; the single incorrect program fails with a runtime error.

[‡]Whether large language models develop implicit world representations (Li et al. 2023; Gurnee and Tegmark 2024), or whether learned models capture the underlying regularities at all—a question open even for other generative models (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 results indicate that the model does not use them to check, 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.

not execute the code; these are reported separately. With these distinctions, the execution contrast holds: CODE_executed solves 20 of 20 mazes, NL 14 and CODE_mental 12; of the eight CODE_mental failures, three are wrong paths and five are responses without a path. The difference between NL and CODE_mental is not supported.

Condition	Easy	Medium	Hard	Overall	No path
NL	6/7	5/7	3/6	14/20	1
CODE_mental	7/7	3/7	2/6	12/20	5
CODE_executed	7/7	7/7	6/6	20/20	0

Table 5. Maze accuracy on Gemini 2.5 Flash (20 mazes; up to three attempts per condition with pass/fail feedback; Gemini command-line interface, non-interactive). One NL success obtained by executing code through the interface is excluded. Responses without a path, in which the model stated that it could not execute code, are counted as failures and reported in the last column.

4.2 Experiment 2: Feedback and Medium in a Repair Loop

Part	Comparison	Success at attempt 1	Success within 3 attempts
2A	Pass/fail feedback (n = 30)	—	8/30 (27%)
	Diagnostic feedback (n = 30)	—	18/30 (60%)
	Single-shot level, all instances: 36/86 (42%); McNemar $p = 0.013$		
2B	Prose schedule (NL, n = 200)	9 (4%)	39 (20%)
	Clingo encoding (ASP, n = 200)	82 (41%)	157 (78%)
	Difference within 3 attempts $+0.59$ (95% CI $+0.52$ to $+0.67$); McNemar $p < 0.0001$		

Table 6. Experiment 2 on Claude Haiku 4.5 without extended thinking (pre-registered). In 2A, the 30 instances whose first encoding ran but produced an invalid schedule branch into two arms that differ only in the content of the feedback. In 2B, every instance is solved in both media under the same loop.

Experiment 2A: interpretable feedback. Of 86 first attempts, 36 were valid (42%: the single-shot, execute-and-check level), 20 did not run (syntax errors) and 30 ran but failed (29 invalid schedules, one unsatisfiable encoding) and branched. Within three attempts the diagnostic arm repaired 18 of 30 instances (60%, 95% CI 42–75%) and the pass/fail arm 8 of 30 (27%, 14–44%); McNemar $p = 0.013$ (Table 6). On attempts to success, the one-sided Wilcoxon test gives $p = 0.0016$, with a Hodges–Lehmann estimate of half an attempt, exactly at the pre-registered threshold (bootstrap 95% interval 0 to 1 attempt): the decision rule is met. The same holds when the unsatisfiable instance is excluded.

Experiment 2B: the same loop in two media. Over 200 paired instances, the ASP encoding was solved within three attempts in 78% of cases and the prose schedule in 20% (difference $+0.59$, 95% CI $+0.52$ to $+0.67$; $p < 0.0001$). At the first attempt, which includes the delegation of the search to Clingo, the rates are 41% and 4%. Among the

113 instances that failed the first attempt in both media, the loop repaired 63% in ASP and 14% in prose. The pre-registered decision rule is met.

How the repair proceeds (exploratory). This analysis was not pre-registered. For each repair step we compare the violations the checker reports before and after the step, and the artifacts themselves (Table 7). The numbers of violations at the first attempt are comparable (median 5 in ASP, 4 in prose). In ASP the model edits its encoding: between consecutive attempts it keeps a median of two thirds of its rules verbatim, against 17% shared by two independent encodings, changes about four statements (additions plus removals) and leaves the instance facts untouched; Clingo then recomputes the schedule, propagating the consequences of the edit. The encoding removes fewer violations per step, but almost never breaks what it had already fixed. In prose the model also starts from the existing schedule and intervenes where it is told—it changes 68% of the jobs involved in a reported violation, against 39% of the others—but it must propagate the consequences of each change itself: half of the start times change at every step, and new violations appear at almost every step.

Per repair step	**ASP**	**NL**
Reported violations removed	38% (233/606)	90% (1,257/1,404)
Steps introducing a new violation	19% (23/118)	90% (325/362)
New violations per step (mean)	0.38	2.86
Repaired violations that return later	13% (3/23)	20% (129/642)
Model's rules kept verbatim (median)	67%	—
Start times kept in the schedule (median)	75%	50%
Reference levels		
Rules shared by two independent encodings (median)	17%	—
Jobs changed: in a reported violation / not	—	68% / 39%

Table 7. How repair proceeds in Experiment 2B (exploratory, not pre-registered). Steps are consecutive attempts of the same instance; for ASP, only steps in which both encodings produced a schedule enter the violation measures.

Two factors, two effects. The two parts of the experiment vary one factor each while holding the loop fixed. In 2A the medium is fixed and the feedback varies: the locality of the edits barely changes (a median of two thirds of the rules kept in both arms), but diagnostic feedback repairs more (60% vs. 27%). In 2B the loop and the feedback are fixed and the medium varies: both the continuity of the solution and the outcome change (78% vs. 20%). Feedback and medium thus act as distinct factors, each with a measurable effect. The medium decides who propagates the consequences of a repair—the executor or the model—and with it the continuity of the solution; by construction it also decides who computes the schedule, and with it the rate at which a single attempt succeeds (41% vs. 4% at the first attempt). The outcome gap therefore cannot be attributed to continuity alone. This is the sense in which Operability is not a description of an execution loop but of the conditions that determine what such a loop achieves.

Return to earlier errors. A repaired violation returns at a later attempt in 13% of cases in ASP and in 20% in prose. In 2A, a violation fixed at the second attempt reappears at the third in 6 of 11 instances with pass/fail feedback and in 3 of 9 with diagnostic feedback. Feedback is taken up, but not always kept (Lu et al. 2025): the loop conditions the next attempt; it does not constrain it.

Two modes of division of labor. In the maze, the model writes a procedure that the interpreter executes: the model is the author of the procedure, the interpreter computes the path. In scheduling, the model writes constraints that Clingo solves: the model is the formalizer, and the KR system computes the solution. In both, the executor computes the solution; they differ in what the model writes. This is why the contrasts of Experiment 2 include the delegation of the search by construction. The repair rates on the instances that failed the first attempt in both media (63% vs. 14%) are close to what two further independent attempts would yield at the first-attempt rates (65% and 9%): they show how often the loop recovers a failed answer, not that repairing beats starting again, which would require a resampling arm that this design does not include.

Symbolic bottleneck. The cost of the formal medium is visible at the first attempt: 20 of 86 encodings in 2A and 52 of 200 in 2B did not run. In 2B these failures are looped like any other, and the ASP advantage holds with them counted.

4.3 Experiment 3: Thinking-Augmented Models

On the two thinking-augmented models, accuracy alone becomes uninformative, and the probe becomes descriptive. The two models exhibit qualitatively distinct behaviors, which we report as such.

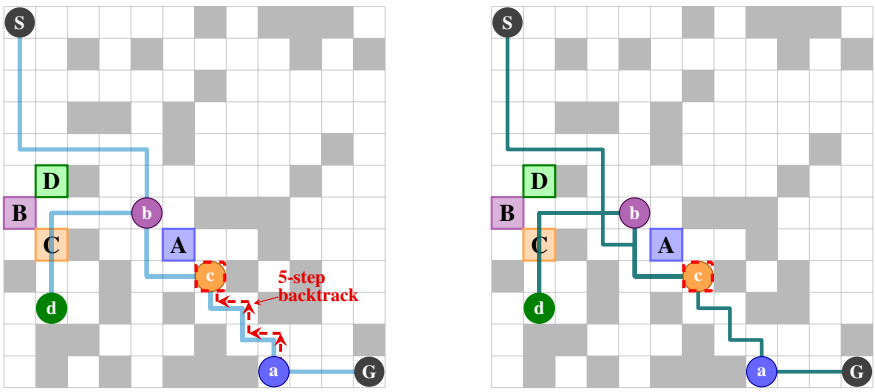
Gemini 3 Flash: path efficiency and unfaithful trace. Gemini 3 Flash achieves 100% accuracy across all three conditions on the harder protocol (Table 8), yet path efficiency differs: NL produces valid but longer paths (1.301 on average), CODE_mental nearly optimal ones (1.042), and CODE_executed optimal ones (1.000).

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 8. Maze results on Gemini 3 Flash Preview, harder protocol (10 trials per condition, 12×12 with 4 keys, adversarial filtering). Accuracy saturates; differences appear in path efficiency.

The traces suggest that the efficiency gradient does not reflect an internalization of the verification loop. In the traces we inspected, 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—appears here also in the extended-thinking regime. The CoT verbalizes one algorithm and executes another. Figure 3 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. 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 model does not detect the discrepancy: 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: no answer within the time limit. Sonnet 4.6 exhibits a qualitatively distinct behavior (Table 9). On CODE_mental, the model fails on 10 of 10 trials—not by producing wrong answers, but by producing no answer within the harness time limit of 350 s; two NL trials end in the same way. The run was made through the Claude command-line interface with its default configuration, and only one full stream was saved.

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

Table 9. Maze results on Claude Sonnet 4.6, harder protocol. †2/10 NL trials produced no answer within the time limit; the remaining 8 produced valid paths, and the reported efficiency averages over those 8 trials. ‡No CODE_mental trial produced an answer within the time limit; efficiency is undefined.

A single CODE_mental trial that we allowed to run without the time limit shows what happens when the model is given more time (Box 1); the full stream log is included in the accompanying repository.

Box 1. Substrate exhaustion on one Sonnet 4.6 trial.
On the one CODE_mental trial run without the time limit, 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 pattern described by Turpin et al. (2023): not only does the verbalized chain-of-thought falsify the underlying computation (BFS is declared, manual distance calculation is executed), the model does not align even its own stated plan with what it then does. The failure mode is the dual of the case in Figure 3: 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. Within the time limit used for the ten trials of Table 9, the model had not reached this point.

Two ways of not executing the artifact. In both models, the written artifact is not executed within the model's own inference. Gemini 3 Flash reports the output of a different process and emits a narrative trace decoupled from the semantics of the code it wrote; Sonnet 4.6, in the one trial run without a time limit, attempts a faithful manual computation and exhausts its output budget without converging. In the two models probed, extended thinking did not act as an internal executor of the model's own code, even when the model could write a correct program.

Across the five models of Experiments 1 and 3, executing the artifact outside the model is what separates the conditions: it solves 45 of 45, 20 of 20 and 19 of 20 mazes in Experiment 1, yields optimal paths on Gemini 3 Flash, and is the only condition in which Sonnet 4.6 answered every trial. The non-executed conditions differ from model to model in how they fall short—lower accuracy, longer paths, or no answer.

5 Discussion and Limitations

5.1 Answering the Research Questions

RQ1: Interpretable feedback. At an equal number of attempts, telling the model which constraints its encoding violates repairs more than telling it only that the schedule is invalid: 60% vs. 27% within three attempts, from a single-shot level of 42% (Experiment 2A). The content of the feedback matters in this setting. Our result is consistent with the planning domain of Stechly et al. (2025), where first-error feedback outperformed a binary verdict (87% vs. 60%), rather than in contrast with their overall conclusion that richer critique adds little. A candidate explanation, which we state as a hypothesis, is that knowing what failed matters most where the artifact can be repaired locally—a plan, an encoding. The feedback does not act as a constraint: diagnostic feedback did not change how locally the model edits its encoding, and repaired violations sometimes return.

RQ2: Syntax vs. execution. Writing the procedure as code does not, by itself, improve on prose: the difference is not significant on Claude Sonnet ($p = 0.21$), it is not supported on Gemini 2.5 Flash, and on Haiku 4.5 both conditions are at floor. Executing the code does make the difference, on all three models: 45 of 45, 20 of 20 and 19 of 20 mazes. The models write correct programs and do not report their output (Section 4.1). On the maze, this advantage is the delegation of the search to the interpreter, obtained at the first attempt. On thinking-augmented models, Gemini 3 Flash's mentally traced code yields nearly optimal paths (1.042), prose longer ones (1.301), and executed code optimal ones (1.000): the written artifact provides some scaffolding, but its reported output is not its execution. Syntax is a precondition for Operability, since it makes artifacts executable; it is not, by itself, a source of grounding.

RQ3: The medium under an identical loop. With the same loop, the same feedback and the same number of attempts, an executable encoding is solved far more often than a prose schedule (78% vs. 20%; Experiment 2B). The medium differs in more than continuity: it also decides who computes the schedule, so the outcome gap is not attributable to continuity alone. What the exploratory analysis shows is how repair proceeds. In ASP the model builds on its previous encoding—two thirds of its rules kept, against 17% shared by independent encodings—and the solver propagates the consequences of each edit; in prose the model repairs where it is told but must propagate the consequences itself, and new violations appear at almost every step. This is what we mean by appropriation; semantic persistence, in the sense of Definition 1, is the condition that separates the two media. Together with RQ1, the experiment also shows that persistence is not sufficient on its own: in Experiment 2A the continuity of the edits is the same with both kinds of feedback, and only diagnostic feedback turns it into repairs.

Extended thinking. Experiment 3 asks whether extended thinking acts as an internal executor of the model's own code, which would realize panel (d) of Figure 1. In the two models probed it did not: Gemini 3 Flash declares one algorithm and reports the output of another; Sonnet 4.6 produced no answer within the time limit and, in the one trial run without it, did not converge. This also makes the framework falsifiable rather than tautological. Operability admits several instantiations of the mechanism that realizes its

conditions, and an internal closed loop is one of them; whether a given model realizes it is an empirical question, which these two probes answer in the negative for these models, on this task.

Division of labor and other formalisms. Our two domains illustrate two modes of division of labor between model and KR system: in the maze, the model writes a procedure that the interpreter executes; in scheduling, the model formalizes constraints that Clingo solves. Both satisfy the three conditions, and in both the executor computes the solution; they differ in what the model writes—a procedure or a specification. A direct test of whether the conditions, rather than the division of labor, are the relevant dimension would require a domain where the 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 10 maps the three conditions onto three KR formalisms; whether the pattern holds for them remains an open empirical question.

The causal taxonomy of Section 3 is not tested by these experiments: on our tasks its categories cannot be separated in the observed failures. Testing whether Structural, Epistemic and Training-induced hallucinations respond differently to Operability requires tasks in which they can be told apart.

	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 10. 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, the model's erroneous artifacts become repairable, and successive repairs build on one another. 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 provide initial evidence on two domains: confirmatory results on one model (Claude Haiku 4.5) for scheduling with an ASP solver; the execution contrast on maze-solving on three models from two families; and exploratory probes on two thinking-augmented models. We note one operational caveat: the symbolic bottleneck observed in Experiment 2 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 Bounded Computation and Repair

A more parsimonious account of the failures we observe is that, within a bounded inference budget, the model cannot carry out a search over a large state space. We adopt it. The data fit it: without execution, success on the maze falls as the state space grows (Table 3); the executed conditions succeed at the first attempt; and stronger models solve the scheduling instances at the first attempt (Section 3.5). The models also write correct programs that they do not carry out (Section 4.1). The failures of the scheduling encodings are of another kind: errors in formalizing a generic program—most often an encoding that lets two jobs overlap on a machine—rather than computations that exceed a budget; the bounded-computation account explains why the model needs an executor, not why its encodings fail. What the bounded-computation account does not by itself address, and what this paper studies, is what happens to the answer the model produces when the computation exceeds its budget. In our experiments that answer is complete, often impossible, and never declined, and whether it can be repaired depends on the conditions of the loop: on the content of the feedback (Experiment 2A) and on the medium in which the answer is held (Experiment 2B).

5.3 Limitations

Decidable ground truth. The maze and scheduling tasks have unambiguous ground truth by design: measuring whether an answer is repaired 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 and scope. The confirmatory results rest on one model, Claude Haiku 4.5 without extended thinking, in the regime where single-shot solutions fail; stronger models solve these instances at the first attempt. Experiment 2A, with 30 branched instances, can detect a large difference between the feedback conditions, not a small one; Experiment 2B, with 200 paired instances, was designed to detect a difference of 15 percentage points. On the maze, the samples are 45 mazes on Claude Sonnet, 20 on Gemini 2.5 Flash and 20 on Haiku 4.5; difficulty cells contain 6 to 15 mazes and should not be over-interpreted individually. Experiment 3 probes, with 60 trials, whether extended thinking gives rise to an internal analogue of the external verification loop. It is a qualitative exploration of failure modes rather than a statistical replication: the underlying question concerns *why* the outputs differ, and the chain-of-thought, the only window onto thinking-model dynamics that 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 across additional thinking architectures and larger samples is future work.

Design of the repair loop. Experiment 2A ablates one condition, interpretable feedback, while holding executability and persistence fixed; the three conditions are not tested as fully separable factors. Persistence is last-step only: each retry sees the previous artifact and its feedback, not the full history of Definition 1. In Experiment 2B the prose retry receives the previous schedule but not the previous reasoning; whether keeping the derivation changes the result is the question of a context-removal experiment that we pre-registered but did not run (Supplementary Materials). No arm resamples the same prompt without history, the baseline of Stechly et al. (2025): whether repairing beats starting again at a fixed medium is therefore not tested, and the repair rates on instances that failed in both media are close to what further independent attempts would give. The contrast between media includes the delegation of the search by construction, and it was run at a slightly easier difficulty than Experiment 2A, by the floor rule stated in Section 3.5. The analysis of how repair proceeds (Table 7) was not pre-registered.

No option to abstain. The prompts required an answer in a fixed format and did not offer an option to abstain. We therefore do not claim that the models would not abstain if allowed to; we report only that, as prompted, they produced impossible answers and did not decline to answer.

Agentic interfaces in the earlier runs. The Claude Sonnet and Gemini 2.5 Flash maze runs were made through agentic command-line interfaces, with each interface's default reasoning settings (Gemini 2.5 Flash reasoned; for Sonnet the setting was not recorded). We report their effects with the results (tool-call errors, attempts to use unavailable tools, one hidden execution); the cross-model evidence for the execution contrast rests on these runs, while the confirmatory experiments use a hardened harness. In those runs, retries did not carry the previous answer, so semantic persistence was absent by construction.

Case-study evidence in Experiment 3. Figure 3 and Box 1 report single-trial close readings: the trace-fidelity failure on Gemini 3 Flash (seed 5003000) and the substrate exhaustion of the one Sonnet 4.6 trial run without a time limit. They are illustrative rather than statistically representative; for Sonnet 4.6, only that trial's full stream was saved. 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 remain challenging for current LLMs. Factual confabulation (inventing facts about the

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

world), increasingly managed by frontier models through retrieval augmentation and post-training, was not directly tested. We expect 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. Table 10 sketches how the three structural conditions map to these formalisms; 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, such as ASP, often fail in semantic repair specifically (Ren et al. 2025). Our data are consistent with this. On Claude Haiku 4.5, 20 of 86 first-attempt encodings in Experiment 2A and 52 of 200 in Experiment 2B did not run, while on the maze 19 of 20 first-attempt Python programs ran and produced optimal paths; on Claude Sonnet 5, 0 of 46 encodings failed to run. The loop compensates in part—the ASP advantage in Experiment 2B holds with these failures counted—but the capacity to act on feedback in a formalism varies with the model.

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 in which a given model class has a low first-attempt validity rate predicts that the loop will repair less, 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 model can repair its own output locally, each repair building on the last. 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 conditions—executability, semantic persistence, and interpretable feedback—that ensure the meaning of a generated artifact is carried forward in a closed loop. In our experiments, when these conditions were met, the erroneous answers of a probabilistic generator became repairable, and its repairs built on one another. Within this characterization, we posed three research questions and offer the following answers.

RQ1: Interpretable feedback. At an equal number of attempts, diagnostic feedback repaired more failed encodings than a pass/fail verdict (60% vs. 27%), from a single-shot level of 42%. The content of the feedback matters; what it does not do is constrain the next attempt.

RQ2: Syntax vs. execution. Writing code without executing it did not improve on prose in any of the three models tested; executing it did. The models wrote correct programs and did not report their output. Syntax is a precondition for executability, not, by itself, a source of grounding.

RQ3: The medium under an identical loop. Under the same loop and the same feedback, an executable encoding was solved far more often than a prose schedule (78% vs. 20%), a gap that includes the delegation of the search. In an exploratory analysis, the model builds on its encoding while the solver propagates the consequences of each edit; in prose it repairs where it is told but must propagate the consequences itself, and new violations accumulate. Persistence is not sufficient on its own: with the same continuity of edits, only diagnostic feedback turns it into repairs. Extended thinking, in the two models probed, did not act as an internal substitute for the external loop.

If in-context processing conditions subsequent generation (Dherin et al. 2025), what matters is what re-enters the context. Operability names the conditions under which what re-enters lets the model build on its own previous work. 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 evaluation through which a model's erroneous answers become repairable. The symbolic bottleneck observed in Experiment 2 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 is practical. We do not propose to make the model less probabilistic: its errors can be contained by giving its outputs symbolic counterparts in which they can be checked and repaired—where the structural conditions admit it. The qualifying clause—*where the structural conditions admit it*—is what makes Operability

a property of model–mechanism–domain configurations rather than of models alone, and it leads to the open questions below.

Future directions. Several trajectories invite further investigation. The first is persistence of the derivation. In our experiments the prose retry received the previous schedule but not the reasoning that produced it; a pre-registered context-removal experiment (Supplementary Materials) asks whether what survives in the context—the derivation, the artifact, or a description of it—changes the model's ability to continue its own work. The second is cross-formalism validation, with ontological reasoners (OWL/DL) and knowledge graph validators (SHACL) as executors (Table 10). The third is a mechanistic interpretability study of the Operability effect on thinking-augmented architectures. 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 of Experiment 3, and could provide a mechanistic account of the appropriation measured behaviorally here. The fourth concerns the extension of Operability beyond domains with a decidable ground truth. The framework 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 observed in operable domains be transferred to open-ended domains through an intermediate generation step in an operable domain? Addressing this question requires a formal analysis of domain properties that goes beyond our current empirical findings. The last, and most ambitious, concerns long horizons. Our experiments measure appropriation over three attempts. Whether the same conditions govern a model's ability to improve its own work incrementally over many iterations—building on what it has produced rather than regenerating it—is an open question, and one that bears on systems that improve themselves autonomously: a system that improves itself must, at the least, be able to take up and refine its own previous output. We leave this direction to future work.

Supplementary Materials

Code, data and pre-registrations are available at https://github.com/AndoSan84/operability. The folder `experiments` contains the pre-registered experiments—designs, configurations, prompts, raw model responses, analysis scripts and post hoc analyses—with the pre-registration of each module identified by a git tag committed before the calls it governs; the other folders contain the earlier, non-pre-registered maze runs.

References

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

Anthropic (2025) Introducing Claude Haiku 4.5. https://www.anthropic.com/news/claude-haiku-4-5. Accessed 1 October 2026.

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)*.

Gundawar A, Valmeekam K, Verma M and Kambhampati S (2024) Robust planning with compound LLM architectures: An LLM-modulo approach. *arXiv preprint arXiv:2411.14484* .

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.

Jha SK, Jha S, Lincoln P, Bastian ND, Velasquez A, Ewetz R and Neema S (2023) Counterexample guided inductive synthesis using large language models and satisfiability solving. In: *MILCOM 2023 – 2023 IEEE Military Communications Conference*. pp. 944–949.

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.

Kambhampati S, Valmeekam K, Guan L, Verma M, Stechly K, Bhambri S, Saldyt L and Murthy A (2024) Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

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)*.

Lu H, Liu Y, Xu J, Nan G, Yu Y, Chen Z and Wang K (2025) Auditing meta-cognitive hallucinations in reasoning large language models. In: *Advances in Neural Information Processing Systems 38 (NeurIPS)*.

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)*.

Quan X, Valentino M, Carvalho D, Dalal D and Freitas A (2025) PEIRCE: Unifying material and formal reasoning via LLM-driven neuro-symbolic refinement. In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. URL https://aclanthology.org/2025.acl-demo.2/.

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* .

Towell GG and Shavlik JW (1993) Extracting refined rules from knowledge-based neural networks. *Machine Learning* 13(1): 71–101.

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* .

Wang X, Chen Y, Yuan L, Zhang Y, Li Y, Peng H and Ji H (2024) Executable code actions elicit better LLM agents. In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

Yang K, Liu J, Wu J, Yang C, Fung YR, Li S, Huang Z, Cao X, Wang X, Wang Y, Ji H and Zhai C (2024) If LLM is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. *arXiv preprint arXiv:2401.00812* .

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)*.