CHARACTERIZING ABDUCTIVE REASONING FAILURES IN LARGE LANGUAGE MODELS WITH ELENCHOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are highly capable of reasoning from known rules to their consequences, but their ability to reason in the opposite direction – from observed behavior to the rules that produced it – remains poorly understood. Recent work has characterized this transition from observations to hypotheses about underlying rules as an "abductive jump" and argued that it remains challenging for current reasoning systems. To investigate this challenge, we introduce *Elenchos*, a framework for evaluating abductive inference in controlled executable formal environments, instantiated here in a dependently typed λ-calculus kernel. Given a pair of kernels, exactly one of which is sound and the other intentionally mutated, agents interact with both through a black-box interface to determine which of the two kernels is sound and identify the mutations responsible for observed behavioral differences.

Evaluating frontier and mid-tier LLMs on Elenchos reveals a striking dissociation between detecting an alteration and identifying its cause: models frequently determine which kernel is sound and gather probes that uniquely distinguish competing mutations, yet often fail to identify the mutations responsible for the observed behavioral differences. To isolate the mechanisms driving this breakdown, we conduct diagnostic ablation studies on GPT-5.4 (high reasoning). These show that forward simulation of mutated systems is substantially harder than simulation of the sound system, but that simulation accuracy does not predict attribution accuracy during interactive reasoning. Remarkably, when the predicted consequences of competing hypotheses are provided explicitly, the model identifies the responsible mutation with near-perfect accuracy. Together, these results point to two interacting bottlenecks in abductive reasoning: deriving the behavioral consequences of competing hypotheses and integrating those consequences with accumulated evidence during interactive attribution.

1 INTRODUCTION

Large language models (LLMs) excel at forward reasoning: deriving consequences from known rules. Many important tasks, however, require reasoning in the opposite direction: inferring hidden rules or explanatory hypotheses from observed behavior. This form of inference is commonly referred to as *abduction*: given observations and a space of possible explanations, the reasoner seeks a hypothesis that best accounts for what has been observed (Peirce, 1998). Sherlock Holmes provides a familiar example: given a collection of clues, he infers the hidden event or mechanism that would explain them. This contrasts with *deduction*, which derives consequences from known rules, and *induction*, which generalizes from observations.

Recent theoretical work characterizes the transition from observed consequences to the underlying rules that generated them as an "abductive jump" that may remain challenging for current reasoning systems (Zahavy, 2026). Understanding this challenge is increasingly relevant as AI systems move toward more autonomous scientific discovery. Systems such as *The AI Scientist* and Google DeepMind's *Co-Scientist* aim to automate or support substantial parts of the scientific workflow, including hypothesis generation, experimental design, and evaluation (Lu et al., 2026; Gottweis et al., 2026;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

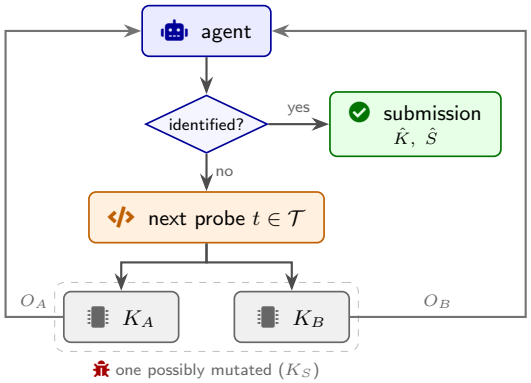


Figure 1: The Elenchos audit loop. *At each iteration, the agent submits a probe to two black-box kernels, K_A and K_B, exactly one of which is corrupted. The kernels return diagnostic tuples, O_A and O_B, that summarize how the probe was processed. These tuples constitute the agent's observations (see Section 3 for details). Using the accumulated observations, the agent assesses whether there is sufficient evidence to identify the corrupted kernel and its active mutation set. If the evidence is insufficient, the agent issues another probe; otherwise, it outputs a hypothesis specifying the corrupted kernel and the inferred mutations.*

Ding et al., 2026). Such systems must not only derive consequences from established hypotheses, but also formulate competing explanations, determine what evidence would distinguish them, and revise them in light of observations.

Here we introduce Elenchos, a controlled setting for studying this abductive jump in formal systems. This framework uses an executable dependently typed λ-calculus kernel (Coquand & Huet, 1988; Martin-Löf, 1984; Church, 1932) and a finite hypothesis space of rule mutations. Agents are presented with a reference kernel and a mutated counterpart and, under a limited probe budget, must infer the mutation configuration responsible for the observed behavioral differences. Because the kernel is executable and the mutation space is known, Elenchos makes it possible to disentangle evidence acquisition from the subsequent identification of the explanation supported by that evidence.

Unlike program-analysis settings that expose source code or internal structure, Elenchos provides only black-box kernel behavior and a curated ontology of possible mutations. Evaluation instances are generated procedurally through compositional rule mutations, yielding a large space of problems without changing the evaluation methodology or requiring a fixed test set (Akhtar & et al., 2026). Although instantiated here in a dependently typed λ-calculus, the framework is substrate-agnostic and can be ported to other formal computational substrates, such as the SKI combinator calculus (Schönfinkel, 1924; Hindley & Seldin, 2008).

Evaluating frontier and mid-tier LLMs on Elenchos reveals a striking *detection–attribution dissociation*: models often identify that a kernel has been altered and generate probes that uniquely isolate the mutation (*detection*), yet fail to infer the underlying rule modification (*attribution*). We therefore complement the main evaluation with diagnostic ablations that separate evidence acquisition, derivation of candidate consequences, and hypothesis–evidence comparison. These experiments reveal that attribution can fail despite conclusive evidence, while making candidate behavioral consequences explicit largely removes the failure. Together, Elenchos and these diagnostic analyses provide a controlled way to characterize where abductive reasoning breaks down in current LLMs.

Related Work. Recent work has studied how language models infer latent rules or explanations from observations, including theoretical and diagnostic analyses of the "abductive jump" (Salimi et al., 2026; Zahavy, 2026; Shi et al., 2026; Balani & Panda, 2026). Related evaluations typically use static completion tasks or synthetic input–output pairs. Elenchos instead studies abduction through interactive probing of an executable formal system, with a bounded hypothesis space and exact ground truth, allowing evidence acquisition and hypothesis attribution to be evaluated separately.

Procedurally generated evaluations have been proposed to reduce contamination and saturation in static benchmarks (Hendrycks et al., 2021; Rein et al., 2023; Akhtar & et al., 2026; Mirzadeh et al., 2024; White et al., 2024). Elenchos applies this principle to the evaluation substrate itself: instances are generated by structured mutations of an executable formal system, so the latent ground truth changes across trials. LLMs have also been used extensively with formal systems as tactic generators or theorem-proving agents (Polu & Sutskever, 2020; Yang et al., 2023; Ren et al., 2025; Hubert et al., 2025), while other work focuses on verification of formal kernels (Sozeau et al., 2020; Carneiro, 2025). Elenchos differs from these settings in treating the formal kernel itself as an un-

known object whose structural integrity must be inferred through black-box interaction (Löh et al., 2010; Altenkirch & Kaposi, 2016; Kovács, 2023).

Elenchos is related to mutation and differential testing, which perturb programs to assess test suites or model robustness (McKeeman, 1998; Yang et al., 2011; Jia & Harman, 2011; Tip et al., 2024; Haroon et al., 2026), and to fault-localization methods that identify buggy program components (Papadakis & Le Traon, 2015; Kang et al., 2024; Yang et al., 2025). It also relates to work on abductive scientific reasoning, such as AI-Noether (Srivastava et al., 2025), and to evaluations of informal abductive reasoning (Bhagavatula et al., 2020; He et al., 2025). In contrast to these settings, Elenchos combines active black-box experimentation, an executable formal substrate, and a bounded space of latent rule mutations with exact verification of the inferred explanation.

2 The Elenchos Framework

The Elenchos framework evaluates an agent's capacity for abductive reasoning by framing system diagnosis as a structural inverse problem. It consists of three core components: (i) a formal system substrate defining the query space, (ii) a mutation ontology defining the space of latent mutation configurations, and (iii) an interactive auditing protocol through which an agent diagnoses those configurations under a limited probe budget.

To isolate this capacity under partial observability, we abstract the underlying rule-governed system independently of any particular formalism. While our primary evaluation instantiates this setup using the dependently typed λ-calculus kernel detailed in Section 3, the general environment is formalized as follows:

Definition 1 (Kernel)**.** *A* kernel K *is a deterministic black-box function*

$$K : \mathcal{T} \to \mathcal{O},$$

where $\mathcal{T}$ denotes the space of probes containing all admissible syntactic terms, and $\mathcal{O}$ represents the space of observable outputs. Let $\mathcal{K}$ denote the set of all such kernels, including the reference kernel K_{ref} and its mutated variants.

We assume that each kernel is induced by an underlying latent rule system that is not directly observable. Mutations operate on this latent representation and induce systematic changes in the observable behavior of the resulting kernel. Agents, however, have access only to black-box interactions with this induced behavior, and must infer the latent structure solely through observed input–output discrepancies.

Definition 2 (Mutation Ontology and Configurations)**.** *A* mutation operator μ *is a transformation over kernels, $\mu : \mathcal{K} \to \mathcal{K}$, that modifies the underlying latent rule structure of a kernel. The mutation ontology is defined as the finite set of these primary operators:*

$$\mathcal{M} = \{\mu_1, \ldots, \mu_n\}.$$

For any mutation configuration set $S \subseteq \mathcal{M}$, we define $K_S \in \mathcal{K}$ as the kernel obtained by applying the mutations in S to the reference kernel K_{ref} under a fixed composition policy.

Composite mutations. Mutations may interact non-linearly, meaning the behavior of a composite configuration K_S cannot generally be predicted from the isolated effects of its constituent mutations. To ensure that mutation attribution is well-defined under black-box access, we restrict the evaluation space to a valid configuration family $\mathcal{F} \subseteq \mathcal{P}(\mathcal{M})$ satisfying a global *identifiability* condition: for $S \in \mathcal{F}$, the mapping

$$S \mapsto K_S$$

is injective over $\mathcal{F}$, ensuring that each induced kernel uniquely determines its underlying mutation configuration in principle.

Elements of $\mathcal{P}(\mathcal{M})$ violating this identifiability condition are excluded from $\mathcal{F}$. Let $k = |S|$ denote the number of active mutations. For $k \geq 2$, excluded cases include: (i) *logical collapse*, where the induced kernel produces uniform outputs across the probe space, and (ii) *symptomatic subsumption*, where a configuration induces the same kernel as one of its proper subsets.

The audit loop. During a session, Elenchos evaluates abductive reasoning as an interactive diagnosis problem (Figure 1). Each instance consists of two kernels (K_A, K_B) implementing the same

formal system, exactly one of which is corrupted. The agent is not told which; instead, it interacts with both through a restricted query interface, submitting probes $\tau \in \mathcal{T}$ and observing outputs (O_A, O_B). From these observations,the agent must identify the corrupted kernel and attribute its active mutations from the predefined ontology. The interaction terminates when the agent submits its final hypothesis, either on its own or after the probe budget P_{max} is exhausted.

Evaluation metrics. *Kernel identification* V_K is correct if the predicted and true kernel labels match. *Mutation attribution* V_A is correct if the predicted and true mutation sets are identical, i.e., if *all* mutations have been correctly identified. After each session, a score $V \in \{0, 1\}$ is returned with $V = 1$ if both kernel identification and mutation attribution are correct, and $V = 0$ otherwise. In the supplementary material, we additionally report Jaccard index, precision, and recall.

Furthermore, in the system prompt, we provide the metric $E_{\text{ELX}} \in [0, 1.5]$ that the LLM is instructed to use for evaluating its performance, combining correctness with a bonus for efficient probing. It is defined as

$$E_{\text{ELX}}(V, P) = V \left(1 + \frac{1}{2}\left(1 - \left(\frac{P}{P_{\text{max}}}\right)^3\right)\right). \tag{1}$$

The score assigns zero to incorrect submissions and rewards correct solutions with a bonus that depends on the number of probes used. The cubic form was chosen to provide only a mild penalty for exploratory probing while rewarding solutions obtained below the probe budget P_{max}.

The task is abductive, not deductive. The task is not primarily deductive, since success does not consist of proving statements within a fixed logical system, and it is not a standard supervised induction task, since no labeled training examples are provided within the evaluation. Rather, we operationalize abductive inference as the problem of recovering a latent mutation configuration $S \in \mathcal{F}$ that best explains observed behavioral discrepancies between paired kernels under black-box access. This corresponds to selecting a hypothesis from a finite configuration space based on partial observational evidence. This notion of abduction is consistent with its classical formulation in the sense of Peirce (Peirce, 1998), where hypotheses are generated to explain observed phenomena rather than derived from axioms or learned from labeled instances.

3 IMPLEMENTATION

Dependently typed λ-calculus. Here we briefly review the core features of the dependently typed λ-calculus used in Elenchos, see also Supplementary Material F. The set of terms $\mathcal{T}$ (which encompasses both expressions and types) is defined by the grammar:

$$\mathcal{T} ::= x \mid \texttt{Type} \mid \lambda x : A.M \mid M\ N \mid \Pi x : A.B, \tag{2}$$

where x denotes a variable, `Type` represents the universe of types, $\lambda x : A.M$ is a function abstraction with domain type A, $M\ N$ is a function application, and $\Pi x : A.B$ is a dependent function type where the result type B may depend on the argument x.

Computation proceeds via β-reduction,

$$(\lambda x : A.M)\ N \longrightarrow_\beta M[x \mapsto N], \tag{3}$$

with $M[x \mapsto N]$ denoting the capture-avoiding substitution of N for x in M. A term is in *β-normal form* when no further reductions are possible. Operationally, a λ-calculus kernel acts as a deterministic type checker, validating terms against a typing context Γ that tracks variable assignments.

Kernel architecture. We instantiate Elenchos using LAMBDAPY, a Python-based implementation of a dependently typed λ-calculus kernel adapted from (Löh et al., 2010). The kernel supports parsing, elaboration, bidirectional type checking, and evaluation, providing a deterministic formal substrate for mutation-based diagnosis. Kernels are equipped with a persistent state that includes the typing context and the evaluated definitions. Details of the surface language are provided in Supplementary Material F.

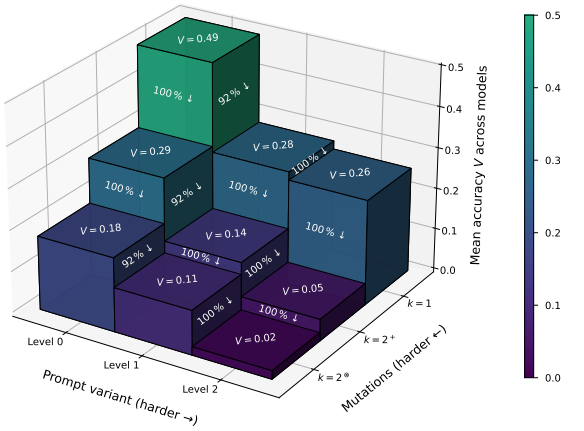


Figure 2: Summary of the results averaged across all evaluated LLMs. *Performance degrades consistently with reductions in system prompt information (Level), higher mutation counts (k), and the presence of interacting mutations $(+ \rightarrow \otimes)$. At the most challenging configuration (minimum prompt information combined with interacting mutations), the average accuracy drops to $V = 0.02$.*

Each kernel execution returns a structured diagnostic tuple: (i) a success indicator specifying whether execution succeeds, (ii) the normalized output value for successful executions (e.g., after applicable β-reductions and other definitional reductions supported by the kernel), and (iii) diagnostic information for failed executions giving the type of error and an error message produced by LAMBDAPY. These outputs implement the observation interface exposed to agents during the audit loop. Three examples of Elenchos' cross-examination procedures are provided in the Supplementary Material E.

Mutation framework. We use a curated ontology of 13 singleton mutations (see Supplementary Material A, Table S1) and construct the corresponding valid configuration family $\mathcal{F}_k$ for $k \leq 2$ according to Supplementary Material A.3. This yields 13 singleton and 10 pairwise configurations from the total of 64 distinguishable pairs, of which three exhibit non-additive interactions $(\otimes)$ and the remaining seven are additive $(+)$ (see Figure S2, Table S3, and Table S4).

Mutations are implemented as configuration-level overrides rather than source-code modifications. Each mutation is represented by a runtime flag, with the default configuration defining the unmutated baseline. This enables compositional, isolated, and reproducible mutation configurations.

Tool use. LLMs may use task-specific tools to submit probes, query the remaining probe budget, inspect the current LAMBDAPY context, and submit an answer. Web search and code execution are disallowed to prevent contamination and automated hypothesis testing or exhaustive search. The evaluation therefore measures abductive attribution from observed behavior without external computational scaffolding, rather than performance achievable with tool augmentation.

The system prompt. We evaluate three conditions that vary the level of procedural guidance provided in the system prompt. These conditions are ordered by increasing difficulty as the amount of auxiliary information is gradually reduced (Levels 0,1,2). All levels expose the full mutation taxonomy (Table S1) and differ only in the operational hints provided. The prompts are included in the Supplementary Material D.

- **Level 0.** The system prompt provides a curated set of canonical probes for each of the 13 mutations. Each probe is designed to produce a behavioral signature specific to its corresponding mutation. Thus, although the agent receives a complete diagnostic reference suite, it must infer which subset of mutations is active in the current instance. The prompt also specifies the exact cardinality k of the active mutation set.
- **Level 1.** The system prompt provides the exact cardinality k of the active mutation set, but omits the canonical example probes.
- **Level 2.** The system prompt informs the agent that there are $k \in \{1, 2\}$ active mutations, but does not provide the exact value of k.

Table 1: **Detection–attribution dissociation, averaged over $k \in \{1, 2\}$ and the three prompt levels (Levels 0–2), with models grouped by family.** *V_K denotes kernel-correct detection; V_A denotes exact mutation-set attribution; and $V_A \mid V_K = 1$ denotes exact attribution conditioned on correct kernel detection. Cells display mean point estimates with asymmetric Wilson 95 % confidence intervals in brackets; the Gap column reports $V_K - V_A$ with paired bootstrap 95 % confidence intervals.*

Model	V_K	V_A	$V_A \mid V_K = 1$	Gap
`ministral-3b-2512`	0.33 [0.29, 0.38]	0.11 [0.08, 0.14]	0.13 [0.08, 0.20]	0.23 [0.17, 0.28]
`ministral-8b-2512`	0.22 [0.18, 0.26]	0.13 [0.10, 0.17]	0.27 [0.18, 0.38]	0.08 [0.03, 0.13]
`ministral-14b-2512`	0.31 [0.26, 0.36]	0.05 [0.03, 0.08]	0.14 [0.09, 0.22]	0.26 [0.21, 0.30]
`mistral-large-2512`	0.65 [0.59, 0.69]	0.31 [0.27, 0.36]	0.38 [0.32, 0.44]	0.33 [0.27, 0.40]
`claude-haiku-4-5`	0.78 [0.74, 0.82]	0.34 [0.29, 0.39]	0.39 [0.34, 0.45]	0.44 [0.38, 0.50]
`gemini-2.5-flash`	0.89 [0.85, 0.92]	0.36 [0.31, 0.41]	0.40 [0.35, 0.46]	0.53 [0.47, 0.58]
`gemini-2.5-pro`	0.93 [0.90, 0.95]	0.35 [0.31, 0.41]	0.38 [0.33, 0.43]	0.57 [0.52, 0.63]
`gpt-5.4-nano`	0.16 [0.13, 0.20]	0.08 [0.05, 0.11]	0.29 [0.18, 0.41]	0.09 [0.05, 0.12]
`gpt-5.4-mini`	0.58 [0.53, 0.63]	0.23 [0.19, 0.28]	0.27 [0.21, 0.34]	0.35 [0.28, 0.41]
`gpt-5.4`	0.86 [0.82, 0.90]	0.37 [0.32, 0.42]	0.42 [0.37, 0.48]	0.50 [0.45, 0.55]
`gpt-5.4_low`	0.94 [0.91, 0.96]	0.41 [0.36, 0.47]	0.44 [0.39, 0.50]	0.52 [0.47, 0.57]
`gpt-5.4_medium`	0.92 [0.89, 0.94]	0.47 [0.42, 0.52]	0.51 [0.46, 0.57]	0.45 [0.40, 0.50]
`gpt-5.4_high`	0.89 [0.85, 0.92]	0.46 [0.41, 0.51]	0.51 [0.46, 0.57]	0.43 [0.38, 0.48]
Overall	0.65 [0.64, 0.66]	0.28 [0.27, 0.30]	0.39 [0.37, 0.41]	0.37 [0.35, 0.38]

4 Experiments and Results

4.1 Experiments over different levels of difficulty

We evaluated 13 LLMs across two mutation counts ($k \in \{1, 2\}$) and three prompt levels (Levels 0, 1, and 2). We set the maximum probe budget to $P_{\max} = 50$. We use 13 singleton mutations and 10 pair-wise mutations out of 64 pairs that are possible to distinguish, three of which exhibit non-additive interaction effects ($k = 2^{\otimes}$) (see Table S4). Each prompt-level and mutation configuration combination is evaluated over 5 independent repetitions. The results are shown in Table 1, Figure 2, Figure 3 and Figure S3.

Performance gradient. As difficulty increases, accuracies decrease, driven by reduced system prompt information and increasing mutation complexity (Figure 2). This gradient suggests that the difficulty scales as intended: tasks remain achievable under favorable conditions, while progressively challenging frontier models. In the most challenging configuration (minimal prompt information with interacting mutations), average accuracy drops to $V = 0.02$.

The detection–attribution gap. Averaged across all 13 LLMs and over $k \in \{1, 2\}$ and prompt levels, models achieve a kernel detection accuracy of $V_K = 0.65$, but an exact mutation attribution accuracy of only $V_A = 0.28$, revealing a substantial detection–attribution gap (Table 1). This discrepancy persists across all evaluated models; the largest gaps appear among the highest-performing models (e.g., `gemini-2.5-pro`), while lower-performing models also retain substantial separation (e.g., `ministral-3b-2512`).

System prompt effects. Providing more structural information through Level-0 system prompts yields substantial performance gains over Level-1 and Level-2 prompts. Weaker models fail almost completely without strong prompt scaffolding (Figure 3, Figure S3). For $k = 2$ and conditioned on correct kernel detection ($V_K = 1$), providing mutation-count information shifts attribution failures: models are more likely to hallucinate mutations when k is known and more likely to omit mutations when it is not (Table S7). In other words, when k is known, the model seems to feel compelled to fill all slots, leading to hallucinated mutation identities. When k is not known, the model leaves mutations unreported.

Effects of interacting mutations. Performance decreases substantially when mutations exhibit non-additive interactions (Figure S3 and across prompt levels in Table S5). At $k = 2$ with Level-2 prompts, most models fail completely, with only two exceptions: `gpt-5.4` with $V = 0.07$ (each such cell holds 15 sessions, so individual values are noisy). Notably, the highest reasoning-budget setting achieves zero accuracy in this case. The results using a Level-1 prompt show a similar pat-

tern: performance drops sharply from "additive" to "interacting" mutations with only two exceptions (`gpt-5.4-mini` and `gemini-2.5-pro`).

Per-mutation difficulty. Attribution difficulty is highly heterogeneous across the mutation ontology: some mutation classes are recovered by nearly all models, whereas others remain near the performance floor (Figure S1).

Probe budget limitations. In $86.5\,\%$ of cases where models correctly detected the corrupt kernel but failed exact attribution ($V_K = 1, V_A = 0$), the allocated probe budget remained unused. This suggests that many failures occur before the probe budget becomes a limiting factor, consistent with premature commitment to incorrect hypotheses rather than insufficient interaction opportunities (Table S7).

Baselines comparison. A random mutation-selection baseline over the configurations the prompt admits achieves accuracies of $1/13 \approx 0.077$ at $k = 1$, $1/78 \approx 0.013$ at $k = 2$, and $1/91 \approx 0.011$ at Level 2, where the count is unknown, so most models substantially exceed chance performance. Randomly selecting the corrupted kernel among $\{A, B\}$ yields a $1/2$ baseline accuracy. A naive exhaustive-search baseline is computationally infeasible due to the combinatorial size of the probe space $\mathcal{T}$. Figure 3 shows that the results across all models far exceed chance level.

4.2 Diagnostic analysis: Decomposing the Attribution Failure

The detection–attribution gap establishes that models often fail to identify the underlying mutation, but not whether they failed to obtain sufficient evidence or failed to interpret the evidence they obtained. We therefore reanalyze each session's output trace – the sequence of probes and the corresponding outputs of both kernels K_{ref} and K_S – to determine whether the model's own probes were sufficient to identify the true mutation configuration.

Specifically, we replay every recorded probe under every candidate mutation configuration admitted by the prompt (13 singletons at $k = 1$, 78 unfiltered mutation pairs at $k = 2$, and 78+13=91 configurations for $k \in \{1, 2\}$). For each candidate, we predict the paired responses of both kernels for every recorded probe and retain the candidate only if both responses match the corresponding observations in the trace. The surviving candidates form the final hypothesis set $\mathcal{F}_{\text{final}}$, representing all configurations consistent with the evidence obtained by the model. We call the evidence *unique* when $|\mathcal{F}_{\text{final}}| = 1$ up to behaviorally indistinguishable configurations: some mutations mask others (ID_07 hides ID_01, ID_03, and ID_10; the subsumed pairs in Figure S2), so no probe can separate a masked variant from the true configuration, and surviving configurations that induce the same kernel are merged before counting. A session with an incorrect final attribution and multiple surviving configurations is classified as a *search failure*; if the true configuration is uniquely identified by the trace but the model reports an incorrect attribution, it is classified as an *interpretation failure*. A submission that differs from the truth only by a masked mutation is behaviorally correct but fails exact-match scoring; such sessions are counted in neither failure class (8 of 4,485 sessions, gray in Figure 3).

Figure 3 shows the results. Three findings stand out. First, unique evidence does not guarantee correct attribution: averaged over all sessions of all models, the recorded probes uniquely identified the mutation configuration in 44.6% of sessions, yet attribution failed in 53.7% of these cases. Second, the failure mode depends on capability and difficulty: stronger models and scaffolds fail predominantly by interpretation, weaker ones by search, while interacting mutations ($k{=}2^{\otimes}$) push even the strongest models toward interpretation failure. Third, interpretation failures are overwhelmingly *self-contradictory*: 84.6% of submitted configurations had already been refuted by the agent's own probes, 79.2% of them visibly as output divergences; over all refuted submissions, the refuting probe falls early (median 29% of session length). Refuting evidence also typically recurs among the final probes, arguing against forgetting as the primary explanation. Furthermore, when the evidence is ambiguous ($|\mathcal{F}_{\text{final}}| > 1$), then 67.9% of correct submissions are the *minimal consistent* configuration, i.e. the smallest hypothesis accounting for all observations. Thus, under ambiguity, models tend to favor parsimony rather than guess.

4.3 Ablations: Locating the Failure

We conduct three ablations on `gpt-5.4_high`, whose strong performance provides a stringent test while still yielding sufficient attribution failures for diagnostic analysis. Together, the ablations

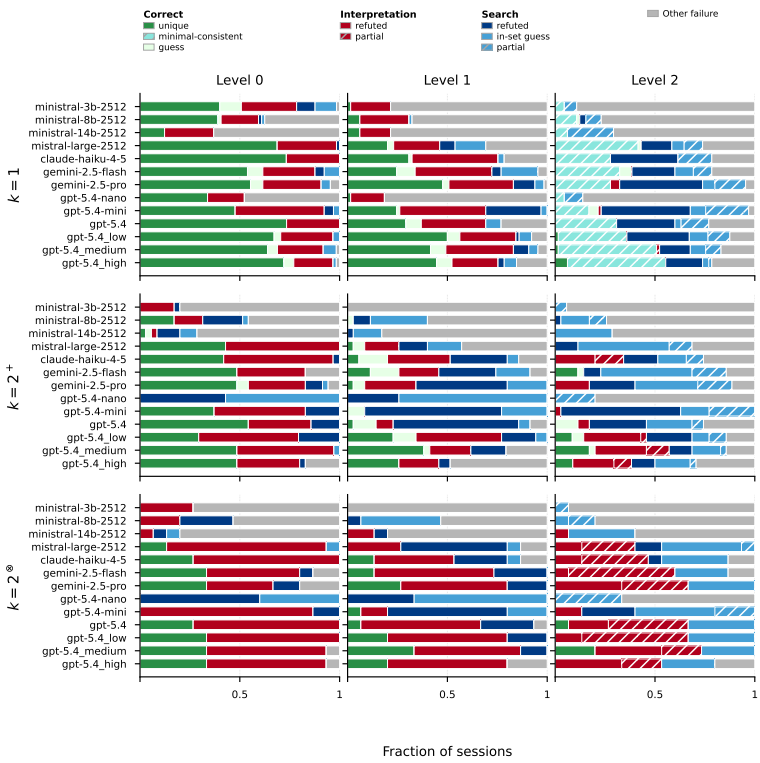


Figure 3: Outcome decomposition per model across mutation order (rows) and prompt scaffolding (columns). Each bar decomposes one model's sessions in one condition by evidence state and attribution outcome (Section 4.2). Green segments denote correct attributions: evidence-unique (solid), minimal-consistent under ambiguous evidence (hatched), and other correct attributions (pale). Interpretation failures are shown in red and search failures in blue; in both categories, the darkest shade marks submissions refuted by the agent's own probe results, while hatching marks partial (subset/superset) attributions. Gray denotes non-verdict outcomes: exhausted budgets, wrong-cardinality submissions, abstentions, and submissions behaviorally identical to the truth.

progressively narrow down the sources of attribution failure and identify which aspects of the reasoning process limit performance. Further details are provided in Supplementary Material C; an overview is shown in Figure S5.

Ablation 1 ("Fresh reader"): commitment is real but minor. Each of the 120 interpretation-failure sessions and 69 originally-correct controls is rebuilt as a read-only dossier (original system prompt plus complete probe log) and re-answered by the same model without probing or access to its original reasoning. Rereading recovers a mean per-session probability $\hat{p}_i = 0.19\ [0.14, 0.24]$ of failures, while retaining a mean 0.87 of controls; 73 of 120 failures never recover in any draw, including 27 of 33 for interacting mutations. Thus, removing the original search context recovers at most a fifth of the failures, indicating that commitment to an early hypothesis cannot account for most interpretation failures.

Ablation 2 ("Informed reader"): supplying candidate effects removes the failure. We tell the model what each candidate configuration would predict for the probes it actually made. The predictions are computed by executing the corresponding kernels; no candidates are eliminated and no verdicts are given, so the model must still compare the predictions with the observed outputs and identify the corrupted kernel. Recovery rises from 0.19 to $0.99\ [0.98, 1.00]$ (paired gain $+0.80$; 98 sessions improve on majority vote and none worsens; exact McNemar $p = 6 \times 10^{-30}$), including 0.99 for interacting mutations. Thus, once the behavioral consequences of candidate configurations are supplied, the model can reliably match them to the observed evidence; deriving those consequences, rather than matching them, is the principal bottleneck.

Ablation 3 ("Forward simulation"): the derivation step degrades with configuration complexity. Given a real probe and a named configuration described only by the free-text mutation catalog, the model predicts the output of that configuration (650 items; 150 further items test the reference kernel). The reference kernel itself is predicted near ceiling (verdict 0.95; exact output in 74 of 75 accepted probes), ruling out basic λ-calculus proficiency as the main limitation. Predicting whether a configuration *diverges* from the reference—the bit used for candidate elimination—falls from 0.87 for singletons to 0.83 for additive and 0.69 for interacting pairs. The dominant error for interacting pairs is a missed divergence: unable to derive a non-additive consequence, the model predicts reference behavior, making the joint effect appear consistent with a single mutation and reproducing

the under-reporting observed in live attribution. Across the 23 configurations we detect no association between simulation accuracy and live attribution accuracy (Spearman $\rho = 0.03$; about 15 live sessions per configuration, so only large effects are detectable). The free-text descriptions therefore support behavioral prediction for most mutation classes but not all (Supplementary Material C), and deriving those predictions becomes increasingly difficult as configurations become more complex.

5 Discussion and Conclusion

We introduced Elenchos, a generative evaluation framework that operationalizes abductive reasoning over formal systems as an interactive structural inverse problem. Across frontier and mid-tier LLMs, we find a persistent detection–attribution dissociation: models often recognize that a system has been altered but struggle to identify the underlying rule modifications. Interacting mutations are particularly difficult to identify in all the LLMs that have been evaluated. Performance drops sharply in these regimes, exposing a lack of robust compositional reasoning when navigating overlapping causal effects. Trace analysis reveals that this reflects two experimentally separable failures: insufficient diagnostic probing and failure to identify the hypothesis supported by the resulting evidence. In 44.6% of sessions, the model's own probes uniquely identified the underlying mutation, yet attribution failed in 53.7% of these cases. Thus, attribution can fail even when the interaction has produced sufficient evidence.

Our diagnostic ablations further localize this failure. Rereading complete probe traces recovers only 19% of attribution failures, whereas supplying the behavioral consequences of candidate configurations raises attribution recovery to 99%. Forward-simulation experiments show that deriving these consequences is itself difficult, particularly for interacting mutations. At the same time, simulation accuracy does not predict attribution difficulty for individual configurations (Spearman $\rho = 0.03$), indicating that attribution is not simply a consequence of successfully simulating the candidate mutations.

A potential concern is that the mutation descriptions themselves may simply be insufficiently informative. Our results argue against this as a sufficient explanation. In the Level-0 condition, the complete mutation catalogue, the number of mutations, and a canonical discriminatory probe for each mutation are provided, yet substantial attribution failures remain. More directly, in the forward-simulation ablation, models predict whether a single-mutation configuration diverges from the reference with 0.87 accuracy. Because this requires deriving the mutation's behavioral consequence from its description, the result shows that the descriptions support substantial reasoning about individual mutation effects. The greater difficulty with mutation combinations, particularly interacting ones, instead points to the challenge of deriving joint behavioral consequences and using them to distinguish competing hypotheses.

Together, these results suggest that attribution involves two interacting demands: deriving the behavioral consequences of competing hypotheses and integrating those consequences with the evidence accumulated during interaction. The findings provide an empirical characterization of the "abductive jump" discussed by Zahavy (2026), while leaving open whether these limitations reflect properties of current LLM architectures or can be overcome through training, representations, or reasoning strategies.

Limitations. Several limitations constrain the scope of our conclusions. First, our empirical evaluation uses a single formal substrate, a dependently typed λ-calculus kernel, although Elenchos itself is substrate-agnostic. Second, the hypothesis space is predefined: the 13 candidate mutations are specified in the system prompt, making the task substantially more constrained than open-ended scientific discovery. Third, performance depends on the prompt and scaffolding used to specify the mutation ontology and task, as illustrated by the substantial differences across prompt levels. Finally, the detailed diagnostic ablations were conducted on `gpt-5.4_high`, whereas the detection–attribution dissociation itself was observed across multiple LLM families. Replication of the mechanistic decomposition across models and substantially different formal environments will therefore be needed to establish its generality.

AI USE STATEMENT

Generative AI tools were used to assist with editing the manuscript text, with figure-plotting and analysis scripts, and with checking internal consistency between sections. All AI-assisted code was reviewed and executed by the authors, and all claims were verified against the experimental logs. Generative AI was not used to generate research ideas, run experiments, or produce the reported results. The authors take responsibility for the final content of this work.

ETHICS STATEMENT

This work does not involve human or animal subjects, personal data, or sensitive datasets, and therefore does not raise issues requiring institutional review or informed consent. All experiments are conducted on procedurally generated instances within a controlled formal system. Although we discuss connections to corrupted computing substrates, Elenchos is an evaluation framework rather than a security auditing or vulnerability-discovery system. The authors report no conflicts of interest that influenced the design, execution, or reporting of this work.

REPRODUCIBILITY STATEMENT

The supplementary archive contains (i) the complete session logs of the main experiment: all 4,485 sessions of the 13 models over the 23 configurations and three prompt levels, one JSON-lines file per session with every probe, both kernels' diagnostic tuples, the final submission, and the ground truth; and (ii) the unmutated LAMBDAPY kernel with its unit tests and a replay script that re-executes each session's probes on the sound kernel and verifies the logged outputs. The system prompts are reproduced in Supplementary Material D, the mutation ontology and admissibility filter in Supplementary Material A, the replay-based failure decomposition in Section 4.2, and the ablation designs in Supplementary Material C. The mutation implementations, the Level-0 probe library, and the analysis code are withheld from the public archive so that the evaluation cannot be memorized by future models; they are available to researchers on request.

REFERENCES

M. Akhtar and et al. When AI benchmarks plateau: A systematic study of benchmark saturation. *arXiv*, 2026. doi: arXiv:2602.16763.

T. Altenkirch and A. Kaposi. Normalisation by Evaluation for Dependent Types. In *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *LIPIcs*, 2016. https://doi.org/10.4230/LIPIcs.FSCD.2016.6.

P. Balani and S. Panda. LLMs Don't Pay for the Jump. *arxiv*, 2026. arXiv:2608.14397.

C. Bhagavatula, R. Le Bras, C. Malaviya, K. Sakaguchi, A. Holtzman, H. Rashkin, D. Downey, W. Yih, and Y. Choi. Abductive Commonsense Reasoning. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1908.05739.

M. Carneiro. Lean4Lean: Verifying a Typechecker for Lean, in Lean. *arXiv*, 2025. https://doi.org/10.48550/arXiv.2403.14064.

A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics, Series 2*, 32(2): 346–366, 1932.

T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76, 1988.

T. Ding, A. Nannapaneni, B. Liu, and L. Zhang. Autonomous Research Agents: A Survey of AI Scientists and the Verification Gap. *arxiv*, 2026. arXiv:2608.05179.

J. Gottweis, W.H. Weng, A. Daryin, and et al. Accelerating scientific discovery with Co-Scientist. *Nature*, 655, 2026. https://doi.org/10.1038/s41586-026-10644-y.

S. Haroon, M.T. Khan, and M.A. Gulzar. Evaluating LLM-Based Test Generation Under Software Evolution. *arXiv*, 2026. arXiv:2603.23443.

Kaiyu He, Peilin Wu, Mian Zhang, Kun Wan, Wentian Zhao, Xinya Du, and Zhiyu Chen. GEAR: A General Evaluation Framework for Abductive Reasoning. *arXiv*, 2025. arXiv:2509.24096.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *NeurIPS*, 2021.

J.R. Hindley and J.P. Seldin. *Lambda-Calculus and Combinators – an Introduction*. Cambridge University Press, 2008.

T. Hubert, A. Banino, and Google DeepMind Theorem Proving Team. Olympiad-Level Formal Mathematical Reasoning with Reinforcement Learning. *Nature*, 2025. https://doi.org/10.1038/s41586-025-09833-y.

Y. Jia and M. Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Trans on Software Engineering*, 37(5), 2011.

S. Kang, G. An, and S. Yoo. A Quantitative and Qualitative Evaluation of LLM-Based Explainable Fault Localization. *Proceedings of the ACM on Software Engineering*, 1(FSE), 2024. https://doi.org/10.1145/3660771.

A. Kovács. smalltt: a Minimal Dependent Type Theory Elaborator. https://github.com/AndrasKovacs/smalltt, 2023. Accessed 2026-05.

A. Löh, C. McBride, and W. Swierstra. A Tutorial Implementation of a Dependently Typed Lambda Calculus. *Fundamenta Informaticae*, 102:177–207, 2010. https://doi.org/10.3233/FI-2010-304.

C. Lu, C. Lu, R.T. Lange, and et al. Towards end-to-end automation of AI research. *Nature*, 651, 2026. https://doi.org/10.1038/s41586-026-10265-5.

P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Naples, 1984. Notes by Giovanni Sambin, Studies in Proof Theory.

W.M. McKeeman. Differential Testing for Software. Digital Technical Journal 10(1), Digital Equipment Corporation, 1998.

I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. *arXiv preprint arXiv:2410.05229*, 2024.

M. Papadakis and Y. Le Traon. Metallaxis-FL: Mutation-Based Fault Localization. *Software Testing, Verification and Reliability*, 25(5-7):605–628, 2015. https://doi.org/10.1002/stvr.1509.

C.S. Peirce. Pragmatism as the Logic of Abduction. In Peirce Edition Project (ed.), *The Essential Peirce, Volume 2*. Indiana University Press, 1998. Originally delivered as a Harvard lecture, 1903.

S. Polu and I. Sutskever. Generative Language Modeling for Automated Theorem Proving. In *arXiv*, 2020. arXiv:2009.03393, https://doi.org/10.48550/arXiv.2009.03393.

D. Rein, B.L. Hou, A.C. Stickland, J. Jackson, S.R. Bowman, et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Z.Z. Ren, Z. Shao, J. Song, H. Xin, H. Wang, W. Zhao, L. Zhang, Z. Fu, Q. Zhu, D. Yang, Z.F. Wu, Z. Gou, S. Ma, H. Tang, Y. Liu, W. Gao, D. Guo, and C. Ruan. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *arXiv*, 2025. arXiv:2504.21801, https://doi.org/10.48550/arXiv.2504.21801.

M. Salimi, S. Adim, D. Parnian, N. Alighardashi, M.J. Siavoshani, and M.H. Rohban. Wiring the 'Why': A Unified Taxonomy and Survey of Abductive Reasoning in LLMs. *arxiv*, 2026. arXiv:2604.08016.

M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92(3–4): 305–316, 1924. doi:10.1007/BF01448013.

D. Shi, X. Li, and J.M. Hernandez-Lobato. When the Canonical Completion Is Wrong: Formalizing and Measuring the Jump in Large Language Models. *arxiv*, 2026. https://arxiv.org/abs/2608.26187.

M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages*, 4(POPL), 2020. https://doi.org/10.1145/3371076.

K. Srivastava, S. Dash, R. Cory-Wright, B. Trager, C. Cornelio, and L. Horesh. Bridging the Gap Between Scientific Laws Derived by AI Systems and Canonical Knowledge via Abductive Inference with AI-Noether. *arxiv*, 2025. https://arxiv.org/abs/2509.23004.

F. Tip, J. Bell, and M. Schäfer. LLMorpheus: Mutation Testing using Large Language Models. *arXiv*, 2024. arXiv:2404.09952, https://doi.org/10.48550/arXiv.2404.09952.

C. White, S. Dooley, M. Roberts, A. Pal, B. Feuer, S. Jain, et al. LiveBench: A Challenging, Contamination-Limited LLM Benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

B. Yang, H. Tian, J. Ren, H. Zhang, J. Klein, T.F. Bissyandé, C. Le Goues, and S. Jin. Integrating Various Software Artifacts for Better LLM-Based Bug Localization and Program Repair. *ACM Transactions on Software Engineering and Methodology*, 2025. arXiv:2412.03905.

K. Yang, A.M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. Prenger, and A. Anandkumar. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. https://doi.org/10.48550/arXiv.2306.15626.

X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In *Proc. 32nd ACM SIGPLAN conf. on Programming language design and implementation - PLDI '11*, pp. 283–294, 2011. https://doi.org/10.1145/1993498.1993532.

T. Zahavy. Position: LLMs Can't Jump. In *Proc of the 43rd Intern Conf on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=klU4737opt.

Supplementary Material

A MUTATION TAXONOMY

Table S1: **Full mutation taxonomy used in the current $|\mathcal{M}| = 13$ implementation.**

ID	Class	Mutation name and description
ID_01	Type Equality	**Equality of Applications Argument** Checker strictly enforces equality on both the function and the argument when comparing applications.
ID_02	Binding Discipline	**Fail to Increment Under Lambda** A substitution intended for an outer binder can incorrectly replace the inner bound variable.
ID_03	Type Equality	**Equality of Bound Indices** `Bound`(0) and `Bound`(1) are considered definitionally equal; dependent types over adjacent binders collapse.
ID_04	Binding Discipline	**Index Shift** `Bound`(i) lookup uses `env`$[\min(i{+}1,\ \mathrm{len}(\texttt{env}){-}1)]$ instead of `env`$[i]$. Causes de Bruijn variable resolution to silently drift by one position.
ID_05	Semantic Environment	**Globals to Zero** Unknown globals normalize to `Zero` rather than staying neutral; type checking proceeds, semantics are junk.
ID_06	Semantic Environment	**Lambda Environment Reversion** The binder is placed at the wrong end of the environment. With the existing `Bound`$(i) \rightarrow$ `env`$[i]$ convention, lambda bodies evaluate against a scrambled scope discipline.
ID_07	Type Equality	**Equality of Inference Terms** Equality of inference terms collapses; all types are judged equal, but inference still succeeds on well-typed terms.
ID_08	Semantic Environment	**Lambda Environment Padding** Closure environment is padded with a neutral element, shifting all argument positions by one.
ID_09	Computational Rule	**Double Application** Function is applied a second time to the result of itself.
ID_10	Type Equality	**Equality of Free Terms** Any two neutral heads become definitionally equal after quotation; unrelated dependent types collapse.
ID_11	Semantic Environment	**Globals to Identity** Unknown globals normalize to the identity function rather than staying neutral.
ID_12	Universe Leveling	**Universe Collapse** `Star`(n) returns `VStar`(n) instead of `VStar`$(n+1)$; the universe hierarchy collapses (`Type`$_n$: `Type`$_n$).
ID_13	Computational Rule	**Argument Dropping** Function argument is dropped; `VStar`(0) (`Type`) is used instead.

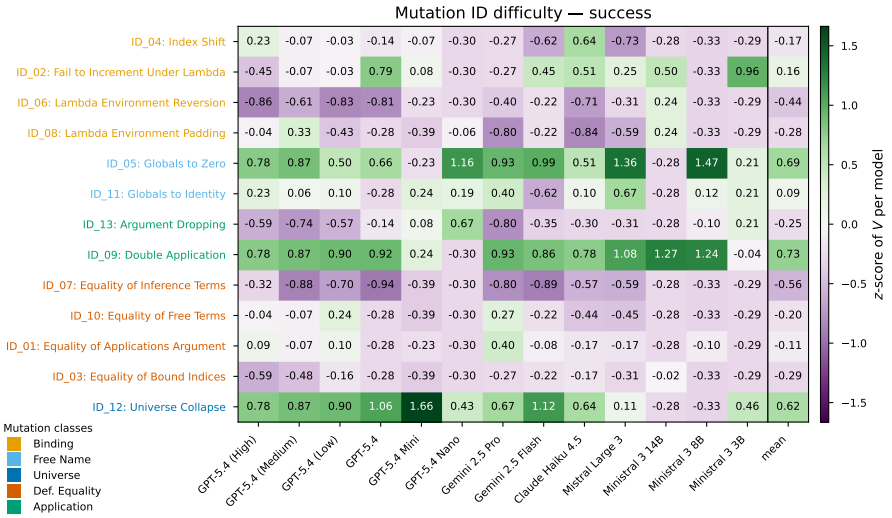
Mutation IDs rank differently in difficulty. There are mutations and classes that are inherently hard or easy regardless of model variants as observed in Figure S1.

A.1 CLASS LABEL MAPPING

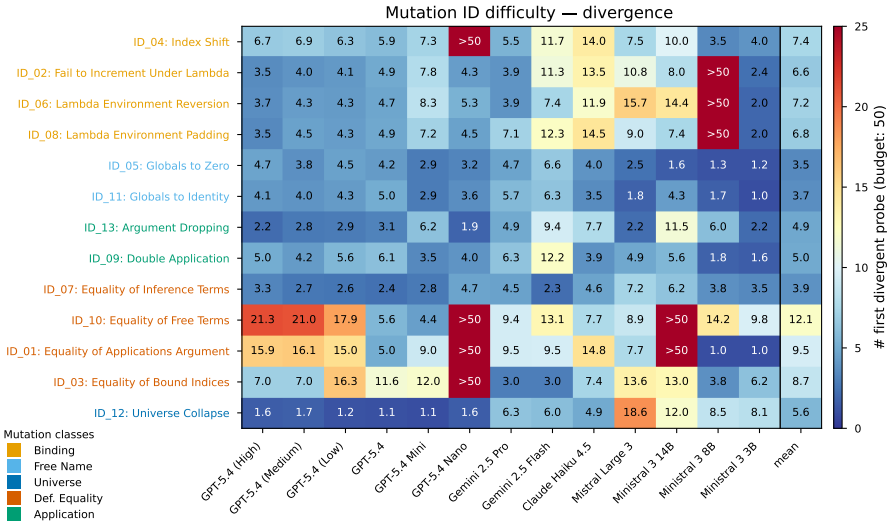
Per-mutation experimental tables (Figure S1) use a coarser secondary grouping — `binding`, `free_name`, `application`, `def_equality`, `universe` — that was generated at experiment time. The five-class taxonomy above (Binding Discipline, Semantic Environment, Computational Rule, Type Equality, Universe Leveling) is the canonical one used by the protocol; the coarser labels are an analysis-time grouping retained for compatibility with the released figures (Table S2 gives the explicit mapping).

A.2 SIMPLE REFLEX AGENT

The `Simple Reflex Agent (RSA)` is designed to diagnose all $n = 13$ mutations defined above (Table S1) assuming singleton mutations $k = 1$. This is achieved by constructing a probe set



(a) Per-mutation exact-match accuracy V, z-scored within each model.



(b) Per-mutation first divergent probe.

Figure S1: **Difficulty of $n = 13$ mutations at order $k = 1$ based on "Ladder" experiment.** Values are averaged across prompt variant conditions. Classes that are consistently hard or easy to determine are shown by striking rows.

Table S2: **Mapping between canonical class names and labels in figures and tables.**

Canonical class	Analysis-time label	Description
Binding Discipline	`binding`	Error in variable resolution such as de Bruijn misalignment and binder shifting
Semantic Environment	`free_name`	Altered substitution or closure environments
Computational Rule	`application`	Corruptions of β-reduction
Type Equality	`def_equality`	Changes to type-level equality or convertability
Universe Leveling	`universe`	Violations of cumulative universe stratification

$\Pi_{\text{RSA}} \subset \mathcal{T}$ such that each pair of distinct mutations exhibits distinguishable behaviour for at least one probe in the set, while the corresponding mutation is also distinguishable from the reference

$$\forall \mu, \mu' \in \mathcal{M}, \mu \neq \mu' \Rightarrow \exists \tau \in \Pi_{\text{RSA}} \,:\, K_{\{\mu\}}(\tau) \neq K_{\{\mu'\}}(\tau) \wedge K_{\{\mu\}}(\tau) \neq K_{\text{ref}}(\tau)\,. \tag{4}$$

By probing both kernels with Π_{RSA}, the agent can uniquely identify the present mutation by comparing kernels output to the expected observation for K_S and K_{ref}.[1]

A.3 Non-degeneracy Filter $k \geq 2$

The candidate family for $k = 2$ is defined by

$$\tilde{\mathcal{F}}_2 = \binom{\mathcal{M}}{2}, \tag{5}$$

which are all possible 2-element combinations of mutations in $\mathcal{M}$. However, not all of them are admissible in the sense that they can be clearly identified. We selected the family of valid combinations $\mathcal{F}_2$ by a deterministic three-condition filter against a curated probe library based.

Predicate. Let $\Pi \subset \mathcal{T}$ denote the probe library: a fixed, ontology-balanced set of LambdaPy source terms that exercises every mutation class. For a $k = 2$ candidate pair $S = \{\mu_i, \mu_j\} \in \tilde{\mathcal{F}}_2$, the divergence vector $\text{div}(S)$ of K_S against K_{ref} on Π is defined as

$$O_S(\tau) = (K_{\text{ref}}(\tau), K_S(\tau), \tau)\,, \tag{6}$$

$$\text{div}(S) = \{O_S(\tau) : \tau \in \Pi\}\ . \tag{7}$$

The pair is *admissible* if

1. **Non-collapse.** $\text{div}(S) \neq \text{div}(\emptyset)$. K_S has to be distinguishable from K_{ref}. Additionally, uniformly rejecting or uniformly accepting pairs are excluded as "total logical collapse": $\exists\, \tau_m, \tau_n \in \Pi$ such that $\tau_m \neq \tau_n \wedge K_S(\tau_m) \neq K_S(\tau_n)$.
2. **Non-subsumption.** $\text{div}(S) \neq \text{div}(\{\mu\})\ \forall\, \mu \in \mathcal{M}$ in particular for μ_i, μ_j. Pairs whose composite divergence is indistinguishable from either singleton are excluded as "symptomatic subsumption" according to Occam's razor.
3. **Identifiability.** $\text{div}(S) \neq \text{div}(S')\ \forall\, S' \in \tilde{\mathcal{F}}_2 \setminus \{S\}$. Two pairs whose divergence vector is indistinguishable are excluded.

Algorithm. For our implementation, we applied the corresponding mutation flag overrides (Section 3) to each of the $\binom{13}{2} = 78$ candidate unordered pairs and ran the Π_{RSA} probe set on both K_{ref} and the merged kernel; each probe under a fresh state which contained only the core in Appendix F. Pairs that fail condition (1) or (2) are rejected. Those failing condition (3) are tested further by extending Π_{RSA} with probes designed specifically to distinguish the pairs. If this is not successful, both pairs are rejected. The remainder is $\mathcal{F}_2$.

Extending to $k > 2$. The algorithm has to be applied iteratively for increasing k, where $\tilde{\mathcal{F}}_k = \binom{\mathcal{M}}{k}$. Only condition (2) needs to be generalized for $S \in \tilde{\mathcal{F}}_k$ to $\text{div}(S) \notin \{\text{div}(S') : S' \in \mathcal{F}_{k'},\ k' < k\}$. The divergence of S must be new, i. e. it must not coincide with the divergence of any set with lower cardinality, in particular to subsets of S.

A.4 Interacting Mutation Sets

We characterized mutation sets S with cardinality $k \geq 2$ as *additive* (highlighted by k^+) if $\forall\, \tau \in \Pi_{\text{RSA}},\ \exists\, S_{\text{sub}} \subsetneq S$ such that $(S_{\text{sub}}$ is additive $\vee\ |S_{\text{sub}}| = 1)\ \wedge\ O_S(\tau) = O_{S_{\text{sub}}}(\tau)$. This basically means, that the behaviour of a corrupted kernel K_S can be fully explained by singleton mutations in S and there are no unexpected results. In contrast, we characterize mutation sets as *interacting* (highlighted by $k^{\otimes}$) if they are not additive. Note that there are different types of interacting sets based on a fixed probe set: They can be partially additive (one mutation is still seen, the other not), deceptive (a singleton mutation is seen that is not active), or causing new behaviour (results can not be assigned to any known mutation).

Cardinalities. We have analyzed our 78 candidate pairs, see Figure S2, and observed three types of mutation sets in our experimental setup, see Table S3.

[1] The list of constructed probes Π_{RSA} and expected output of K_{ref} and K_S can be provided on request.

Table S3: **Types of mutation pairs in Figure S2.**

Type	Colours in Figure S2	Count
Admissible, additive (used at $k = 2^{+}$)	blueish green	55
Admissible, interacting (used at $k = 2^{\otimes}$)	orange, red, purple	9
Total collapse (uniform accept/reject)		0
Subsumed by one of $\{\mu_i, \mu_j\}$	light grey, dark grey	14
Total		78

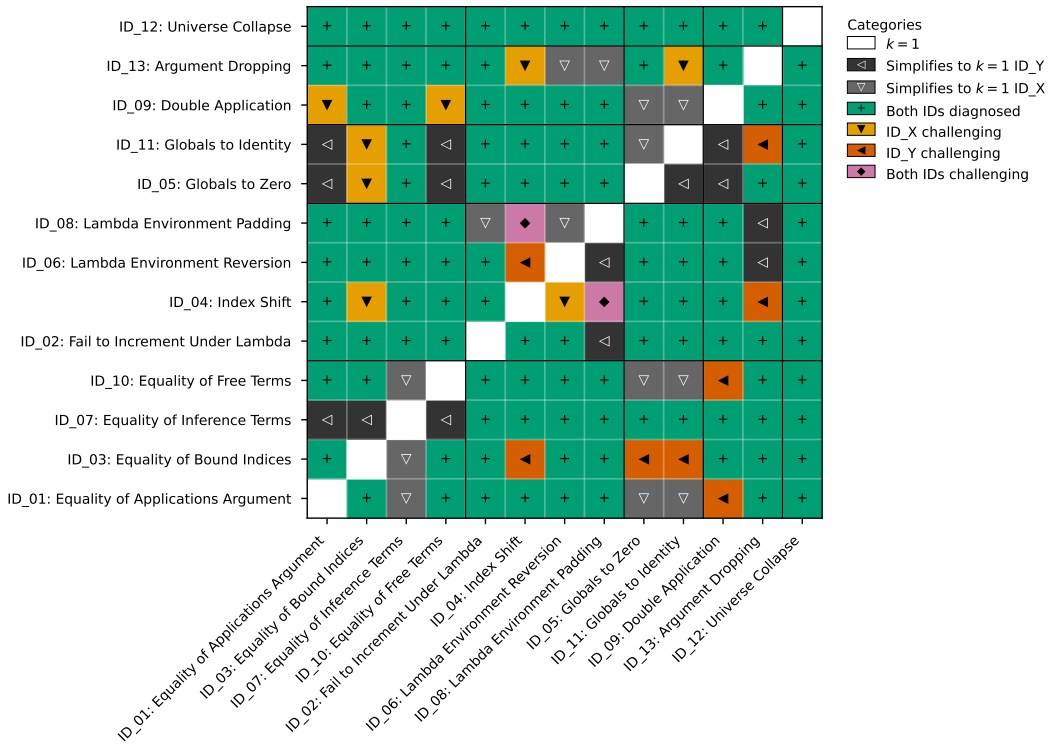


Figure S2: **Interaction scheme of mutations.** Subsumed mutations (grey) are determined by i) indication of implementation ii) indistinguishable behaviour from $k = 1$ mutations when facing $\Pi_{\texttt{RSA}}$. All combinations that are coloured are valid. Blueish green $(+)$ is the easiest type of combination: Both mutations are additive and do not interact. There RSA produces $V = 1$. Two interacting types $(\otimes)$ are observed in our setting: i) Partial observability (orange, red). RSA identifies only one mutation. Combinations kernel behaviour is still distinguishable from all others. ii) Emergence (purple, diamond) where kernel shows a fully interacting behaviour.

A.5 Compound-Mutation Sets

The mutation families used in the ladder experiment are shown in Table S4.

Table S4: **Mutation configurations evaluated in the main experiment: all 13 singletons and the 10 curated pairs (7 additive, 3 interacting).**

Order	Mutation Set	Classes
$k = 1$	{ID_01}	def_equality
	{ID_02}	binding
	{ID_03}	def_equality
	{ID_04}	binding
	{ID_05}	free_name
	{ID_06}	binding
	{ID_07}	def_equality
	{ID_08}	binding
	{ID_09}	application
	{ID_10}	def_equality
	{ID_11}	free_name
	{ID_12}	universe
	{ID_13}	application
$k = 2^+$	{ID_01,ID_03}	def_equality
	{ID_02,ID_04}	binding
	{ID_02,ID_06}	binding
	{ID_03,ID_10}	def_equality
	{ID_04,ID_09}	application × binding
	{ID_06,ID_09}	application × binding
	{ID_06,ID_10}	def_equality × binding
$k = 2^\otimes$	{ID_04,ID_06}	binding
	{ID_04,ID_08}	binding
	{ID_09,ID_10}	def_equality × application

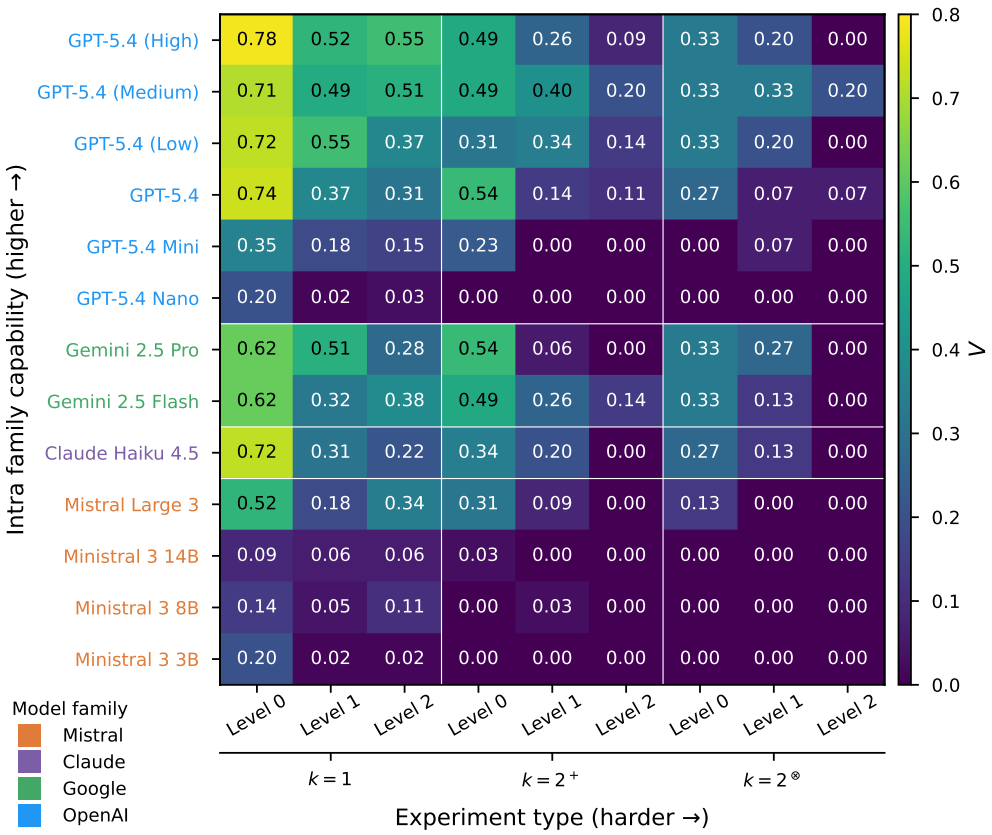


Figure S3: **Performance across experimental configurations ("Ladder" experiment).** *Accuracy increases with model capability but decreases with reduced system prompt information (Levels), increased mutation complexity* (k)*, and non-additive mutation interactions* $(+ \to \otimes)$*. The reported metric is the averaged exact-match accuracy* (V) *across mutation sets, which requires correct identification of whether either kernel is corrupted and, when applicable, the precise attribution of all active mutations.*

B Detailed Model-by-condition Tables

The Section 4 main text reports only the headline inline, e. g. Table 1. All other per-model and per-condition tables are collected here. The wide tables are set on their own landscape pages.

"Ladder" experiment. Table S5 marginalises over scaffold to expose the single vs. double dissociation as well as the interacting difficulty. Similarly, Table S6 summarizes the effect of hint levels in the system prompt per model. Table S7 expands the over/under-reporting breakdown at $k = 2$.

Supporting figures. A visualization of the detection–attribution gap in the "Ladder" experiment is shown in Figure S4.

Table S5: **Per-model comparison of single vs. double vs. interacting mutations.** Marginal accuracy by mutation count k on the "Ladder" experiment: exact correctness V, kernel detection V_K, exact attribution V_A, Jaccard partial credit V_A^J, conditional attribution correctness $V_A|V_K = 1$, and ratio of used probes P/P_{max}. Numbers are mean values over prompt levels and corresponding mutations sets. For $k = 2$, superscripts indicate whether the mutations are additive $(+)$ or interacting $(\otimes)$.

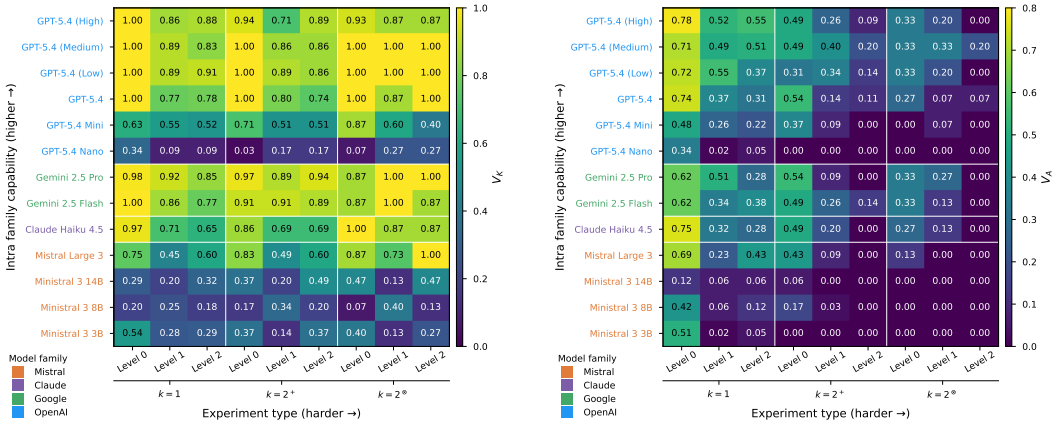
Model	k	n	V	V_K	V_A	V_A^J	$V_A \mid V_K$	P/P_{max}
`ministral-3b-2512`	1	195	0.077	0.369	0.190	0.190	0.208	0.616
	2^+	105	0.000	0.295	0.000	0.094	0.000	0.727
	$2^\otimes$	45	0.000	0.267	0.000	0.015	0.000	0.775
`ministral-8b-2512`	1	195	0.097	0.210	0.200	0.200	0.463	0.966
	2^+	105	0.010	0.238	0.067	0.160	0.040	0.988
	$2^\otimes$	45	0.000	0.200	0.000	0.074	0.000	0.980
`ministral-14b-2512`	1	195	0.072	0.272	0.082	0.085	0.264	0.999
	2^+	105	0.010	0.352	0.019	0.122	0.027	1.000
	$2^\otimes$	45	0.000	0.356	0.000	0.096	0.000	1.000
`mistral-large-2512`	1	195	0.349	0.600	0.451	0.451	0.581	0.573
	2^+	105	0.133	0.638	0.171	0.363	0.209	0.619
	$2^\otimes$	45	0.044	0.867	0.044	0.274	0.051	0.550
`claude-haiku-4-5`	1	195	0.415	0.774	0.451	0.456	0.536	0.763
	2^+	105	0.181	0.743	0.229	0.386	0.244	0.813
	$2^\otimes$	45	0.133	0.911	0.133	0.404	0.146	0.852
`gemini-2.5-flash`	1	195	0.441	0.877	0.446	0.459	0.503	0.378
	2^+	105	0.295	0.905	0.295	0.471	0.326	0.470
	$2^\otimes$	45	0.156	0.911	0.156	0.393	0.171	0.427
`gemini-2.5-pro`	1	195	0.467	0.918	0.467	0.495	0.508	0.342
	2^+	105	0.200	0.933	0.210	0.379	0.214	0.434
	$2^\otimes$	45	0.200	0.956	0.200	0.430	0.209	0.419
`gpt-5.4-nano`	1	195	0.082	0.174	0.133	0.133	0.471	0.153
	2^+	105	0.000	0.124	0.000	0.083	0.000	0.170
	$2^\otimes$	45	0.000	0.200	0.000	0.000	0.000	0.152
`gpt-5.4-mini`	1	195	0.231	0.569	0.318	0.326	0.405	0.304
	2^+	105	0.076	0.581	0.152	0.298	0.131	0.328
	$2^\otimes$	45	0.022	0.622	0.022	0.211	0.036	0.232
`gpt-5.4`	1	195	0.472	0.851	0.472	0.487	0.554	0.310
	2^+	105	0.267	0.848	0.267	0.386	0.315	0.348
	$2^\otimes$	45	0.133	0.956	0.133	0.396	0.140	0.325
`gpt-5.4_low`	1	195	0.549	0.933	0.549	0.574	0.588	0.265
	2^+	105	0.267	0.914	0.267	0.421	0.292	0.347
	$2^\otimes$	45	0.178	1.000	0.178	0.478	0.178	0.295
`gpt-5.4_medium`	1	195	0.569	0.908	0.569	0.585	0.627	0.311
	2^+	105	0.362	0.905	0.362	0.524	0.400	0.449
	$2^\otimes$	45	0.289	1.000	0.289	0.507	0.289	0.404
`gpt-5.4_high`	1	195	0.621	0.913	0.621	0.628	0.680	0.326
	2^+	105	0.276	0.848	0.276	0.400	0.326	0.484
	$2^\otimes$	45	0.178	0.889	0.178	0.419	0.200	0.461

Table S6: **Per-model comparison of scaffolding level.** Marginal accuracy by prompt level on the "Ladder" experiment: exact correctness V, kernel detection V_K, exact attribution V_A, Jaccard partial credit V_A^J, conditional attribution correctness $V_A|V_K = 1$, and ratio of used probes P/P_{max}. Numbers are mean values over mutations sets.

Model	Prompt	n	V	V_K	V_A	V_A^J	$V_A \mid V_K$	P/P_{max}
`ministral-3b-2512`	Level 0	115	0.113	0.470	0.287	0.378	0.241	0.206
	Level 1	115	0.009	0.217	0.009	0.009	0.040	0.874
	Level 2	115	0.009	0.313	0.026	0.026	0.028	0.932
`ministral-8b-2512`	Level 0	115	0.078	0.174	0.287	0.336	0.450	0.936
	Level 1	115	0.035	0.296	0.043	0.078	0.118	0.991
	Level 2	115	0.061	0.183	0.070	0.100	0.333	0.996
`ministral-14b-2512`	Level 0	115	0.061	0.339	0.087	0.130	0.179	1.000
	Level 1	115	0.035	0.191	0.035	0.058	0.182	0.998
	Level 2	115	0.035	0.391	0.035	0.104	0.089	1.000
`mistral-large-2512`	Level 0	115	0.409	0.791	0.539	0.620	0.516	0.331
	Level 1	115	0.130	0.496	0.157	0.223	0.263	0.749
	Level 2	115	0.191	0.652	0.243	0.361	0.293	0.672
`claude-haiku-4-5`	Level 0	115	0.548	0.939	0.609	0.678	0.583	0.679
	Level 1	115	0.252	0.722	0.261	0.339	0.349	0.832
	Level 2	115	0.122	0.687	0.157	0.267	0.177	0.859
`gemini-2.5-flash`	Level 0	115	0.539	0.957	0.539	0.583	0.564	0.172
	Level 1	115	0.278	0.896	0.287	0.359	0.311	0.475
	Level 2	115	0.261	0.817	0.261	0.420	0.319	0.590
`gemini-2.5-pro`	Level 0	115	0.557	0.965	0.557	0.591	0.577	0.161
	Level 1	115	0.339	0.922	0.348	0.429	0.368	0.458
	Level 2	115	0.157	0.896	0.157	0.333	0.175	0.520
`gpt-5.4-nano`	Level 0	115	0.113	0.209	0.191	0.214	0.542	0.111
	Level 1	115	0.009	0.139	0.009	0.061	0.062	0.151
	Level 2	115	0.017	0.139	0.026	0.026	0.125	0.212
`gpt-5.4-mini`	Level 0	115	0.270	0.687	0.383	0.470	0.392	0.141
	Level 1	115	0.113	0.548	0.183	0.246	0.206	0.353
	Level 2	115	0.087	0.504	0.122	0.191	0.172	0.411
`gpt-5.4`	Level 0	115	0.617	1.000	0.617	0.670	0.617	0.176
	Level 1	115	0.261	0.791	0.261	0.328	0.330	0.371
	Level 2	115	0.217	0.800	0.217	0.336	0.272	0.424
`gpt-5.4_low`	Level 0	115	0.548	1.000	0.548	0.629	0.548	0.174
	Level 1	115	0.443	0.904	0.443	0.519	0.490	0.346
	Level 2	115	0.252	0.904	0.252	0.397	0.279	0.362
`gpt-5.4_medium`	Level 0	115	0.591	1.000	0.591	0.667	0.591	0.211
	Level 1	115	0.443	0.896	0.443	0.504	0.495	0.423
	Level 2	115	0.374	0.861	0.374	0.497	0.434	0.462
`gpt-5.4_high`	Level 0	115	0.635	0.974	0.635	0.693	0.652	0.231
	Level 1	115	0.400	0.817	0.400	0.446	0.489	0.446
	Level 2	115	0.339	0.878	0.339	0.455	0.386	0.497

Table S7: **Fault characterization for $k = 2$ double mutation in the "Ladder" experiment.** Per-model and k-hint level prompts, attribution modes are analyzed for n sessions with $V_K = 1$. Each entry shows the proportion of responses that were classified as correct, incomplete (partially correct but missing mutations), mixed (partially correct but containing incorrect mutations), or incorrect. Proportions sum to one for each model and prompt. $< P_{\text{max}}$ gives the proportion of sessions with probe budget NOT exhausted although incorrect mutation attribution $V_K = 1 \wedge V_A = 0$.

Model	Prompt	n	Correct	Incomplete	Mixed	Incorrect	$< P_{\text{max}}$
`ministral-3b-2512`	Level 0-1	26	0.000	0.192	0.192	0.615	0.731
	Level 2	17	0.000	0.000	0.000	1.000	0.118
`ministral-8b-2512`	Level 0-1	25	0.040	0.000	0.680	0.280	0.208
	Level 2	9	0.000	0.778	0.000	0.222	0.222
`ministral-14b-2512`	Level 0-1	29	0.034	0.138	0.483	0.345	0.000
	Level 2	24	0.000	0.625	0.000	0.375	0.000
`mistral-large-2512`	Level 0-1	70	0.229	0.000	0.643	0.129	0.926
	Level 2	36	0.000	0.750	0.000	0.250	0.917
`claude-haiku-4-5`	Level 0-1	82	0.305	0.000	0.573	0.122	0.965
	Level 2	37	0.000	0.541	0.135	0.324	0.946
`gemini-2.5-flash`	Level 0-1	92	0.359	0.000	0.435	0.207	0.898
	Level 2	44	0.114	0.705	0.023	0.159	0.923
`gemini-2.5-pro`	Level 0-1	93	0.323	0.000	0.430	0.247	0.889
	Level 2	48	0.000	0.479	0.208	0.312	0.979
`gpt-5.4-nano`	Level 0-1	12	0.000	0.000	0.417	0.583	1.000
	Level 2	10	0.000	0.000	0.000	1.000	1.000
`gpt-5.4-mini`	Level 0-1	65	0.138	0.000	0.569	0.292	1.000
	Level 2	24	0.000	0.292	0.083	0.625	0.958
`gpt-5.4`	Level 0-1	91	0.319	0.000	0.451	0.231	1.000
	Level 2	41	0.122	0.390	0.195	0.293	1.000
`gpt-5.4_low`	Level 0-1	96	0.323	0.000	0.552	0.125	1.000
	Level 2	45	0.111	0.444	0.111	0.333	1.000
`gpt-5.4_medium`	Level 0-1	95	0.432	0.000	0.484	0.084	0.981
	Level 2	45	0.222	0.378	0.178	0.222	0.971
`gpt-5.4_high`	Level 0-1	85	0.400	0.000	0.424	0.176	0.922
	Level 2	44	0.068	0.386	0.227	0.318	0.976
Overall	Level 0–2	1285	0.216	0.163	0.370	0.251	0.865



(a) Kernel accuracy V_K.

(b) Mutation accuracy V_A.

Figure S4: **Per-condition accuracy heatmaps in the "Ladder" experiment of** 13 **models.** The Elements configurations of (model, prompt variant, mutation order). The resulting matrices in (a) and (b) are visibly different. While the monotonicity of kernel accuracy (a) is unclear because the models' performance saturates early, the mutation accuracy (b) shows the canonical ladders for capability and task difficulty, as seen in Figure S3.

C Ablations: Designs and Full Results

This appendix gives the designs and full results of the three ablations summarized in Section 4.3, all run on `gpt-5.4_high`; the sessions, failure categories, and evidence-uniqueness criterion are those of Section 4.2.

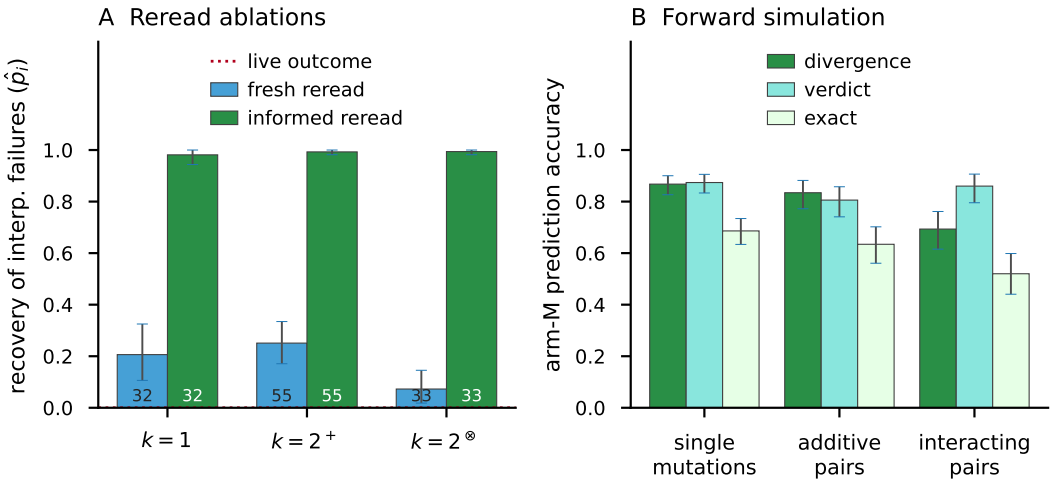


Figure S5: Ablations on `gpt-5.4_high`. A: mean per-session recovery $\hat{p}_i$ of the 120 interpretation-failure sessions when the model rereads the transcript without the search context (fresh) and with the computed effects of every candidate supplied (informed); session-level bootstrap 95 % intervals, session counts at the bar base. **B**: forward-simulation accuracy by configuration bucket at three nested tiers (divergence from the reference, accept/reject verdict, exact output), Wilson 95 % intervals, N=650 items.

C.1 Ablation 1 (Fresh Reader): Rereading the Evidence Without the Search Context

Interpretation failures could have a benign cause: the interactive search context itself. An agent that formed an early hypothesis while probing may remain committed to it, so its failure would reflect anchoring rather than an inability to read the evidence. The fresh-reader experiment separates these. For each session with uniquely determining evidence we rebuild a read-only dossier: the session's original system prompt verbatim, followed by the complete probe log exactly as the live agent received it, followed by a single-answer contract. The same model then answers in one completion, without the ability to probe and without access to its original reasoning. If interpretation failures stem from commitment, a fresh read of identical evidence should recover them; if they stem from the reading itself, they should persist. We evaluate all 120 interpretation-failure sessions and, as controls, a sample of 69 of the 114 originally-correct sessions with unique evidence, matched on mutation order, with five independent reread draws per failure and three per control (807 completions), and report per-session recovery probabilities $\hat{p}_i$ with session-level bootstrap intervals.

Findings. Commitment is real but minor (Figure S6). Rereading recovers a mean $\hat{p}_i = 0.19$ $[0.14, 0.24]$ of interpretation failures, while control sessions are retained at 0.87 $[0.80, 0.93]$. Fewer than 20 % of erroneous attributions are therefore explained by the search context; the remainder persist under ideal reading conditions, with the evidence complete, conclusive, and re-presented without any interactive history. Recovery is stochastic rather than concentrated: across five draws only 3 of 120 failures recover reliably, 44 recover in some draws, and 73 never recover. It is weakest precisely where attribution is hardest: for interacting mutations, 27 of 33 failure sessions never recover in any draw, and at prompt Levels 0 and 2 fewer than 4 % of reread draws succeed. Two secondary observations qualify the picture. Control retention degrades without scaffolding (from $\geq$0.94 under Level-0 prompts to $\approx$0.5–0.6 in the weakest cells), so the interactive context carries useful state as well as harmful commitment; fresh reading is not uniformly superior. And because rereads are stochastic, a single-read critic would under-estimate the recoverable fraction; majority

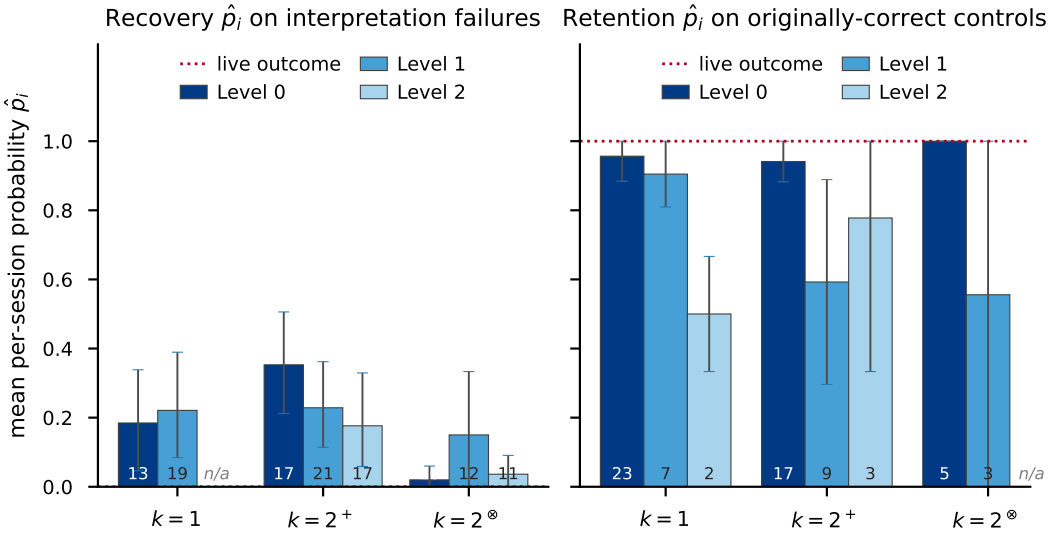


Figure S6: Fresh-reader experiment: rereading recovers only a minority of interpretation failures. All 120 interpretation-failure sessions of `gpt-5.4_high` with uniquely determining evidence, and a matched sample of 69 originally-correct sessions, are rebuilt as read-only dossiers (original system prompt plus the exact probe log) and re-answered by the same model without probing or access to its original reasoning; bars give the mean per-session recovery probability $\hat{p}_i$ over five independent reread draws (three for controls), with session-level bootstrap 95 % intervals and per-cell session counts at the bar base. **Left**: on interpretation failures (live attribution wrong by definition, dotted line), rereading recovers $\hat{p}_i = 0.19\ [0.14, 0.24]$ overall; recovery is near zero for interacting mutations ($k{=}2^\otimes$; 27 of 33 sessions never recover in any draw). **Right**: originally-correct controls are retained at 0.87 overall, but retention degrades with weaker scaffolding, indicating that the interactive search context carries useful state in addition to harmful commitment. Cells marked *n/a* contain no qualifying sessions.

voting over several independent reads is the natural mitigation. What the fresh reader leaves open is where the persistent 80 % fail. Reading a transcript against the mutation catalog requires two steps: deriving what each candidate configuration would do on the recorded probes, and matching those predictions to the observed outputs. The next ablation separates them.

C.2 Ablation 2 (Informed Reader): Supplying the Behavioral Effects of Candidate Configurations

The informed-reader experiment removes the derivation step. Every session of the fresh-reader experiment is rebuilt once more, now with an *expected behavioral effects* section inserted between the system prompt and the probe log. To construct it we replay the session's probe sequence, in order and from a fresh kernel, against every configuration admitted by the prompt (13 singletons, 78 pairs, or 91 at Level 2). A probe is *selective* if the candidates do not all predict the same observation; non-selective probes carry no information and are omitted. For each selective probe the section lists the reference kernel's prediction once, followed by the groups of configurations that share an identical deviating prediction; configurations not listed behave like the reference on that probe. The model still has to do the matching itself, including which kernel is corrupted: the dossier contains no eliminations, no mismatch lists, and no per-candidate verdicts. The 189 sessions and the draw counts are identical to the fresh-reader experiment, so the two conditions are paired per session, and together with the live session they form a three-condition sequence on identical evidence: live (search, derive, match), fresh reread (derive, match), informed reread (match only).

Findings. Supplying the effects removes the failure almost entirely (Figure S7). On the 120 interpretation-failure sessions, mean recovery rises from $\hat{p}_i = 0.19\ [0.14, 0.24]$ under fresh rereading to $0.99\ [0.98, 1.00]$ under informed rereading, a paired gain of $+0.80\ [+0.75, +0.85]$; on per-session majority votes, 98 sessions improve and none worsens (exact McNemar $p = 6 \times 10^{-30}$). The gain is largest where fresh rereading was weakest: for interacting mutations recovery rises from 0.07 to

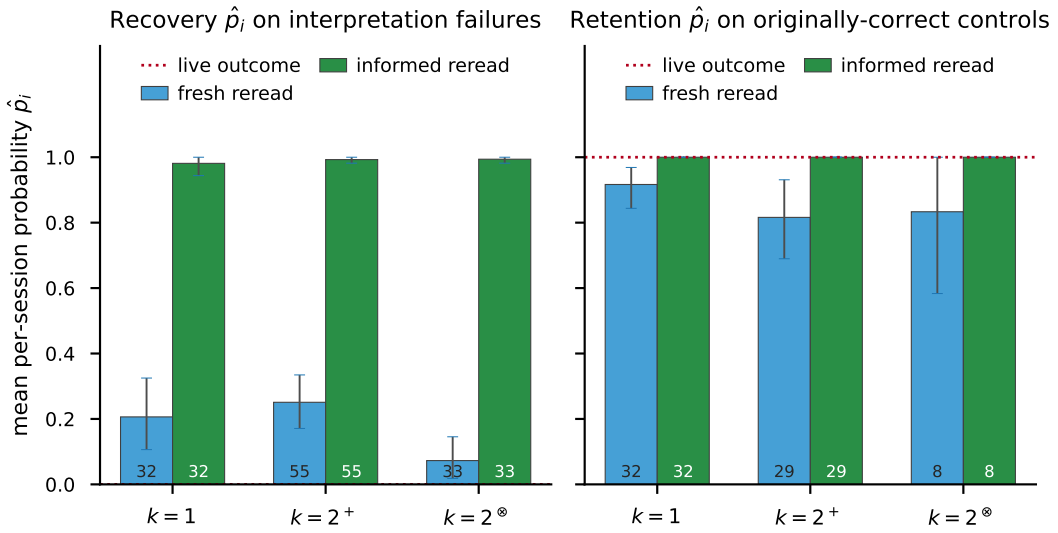


Figure S7: Informed-reader ablation: supplying the computed effects of every candidate configuration removes the attribution failure. The 189 sessions of the fresh-reader experiment are reread once more with an expected-behavioral-effects section added to the dossier (each candidate's kernel-computed output on the session's discriminating probes; no eliminations or verdicts), with the same draw counts, so fresh and informed rereads are paired per session. Bars give the mean per-session recovery probability $\hat{p}_i$ with session-level bootstrap 95 % intervals and per-cell session counts at the bar base; dotted lines mark the live outcome (0 on failures, 1 on controls, by definition). **Left**: on interpretation failures, recovery rises from 0.19 to 0.99 (paired $+0.80$ $[+0.75, +0.85]$; 98 sessions improve on majority vote, none worsens), with the largest gain on interacting mutations ($k{=}2^\otimes$: 0.07 $\rightarrow$ 0.99). **Right**: originally-correct controls are retained at 1.00 under informed rereading, so the added material does not disturb correct readings.

0.99 ($+0.92$ $[+0.85, +0.98]$), for additive pairs from 0.25 to 0.99, and for singletons from 0.21 to 0.98. No session is unrecoverable: every one of the 120 failures recovers in at least two of its five draws, 119 in a majority and 116 in every draw, including the 73 that never recovered under fresh rereading; recovery is also uniform across prompt levels (every cell $\geq$0.97), which fresh rereading was not. Control sessions are not harmed by the additional material; their retention rises from 0.87 to 1.00 ($+0.13$ $[+0.07, +0.21]$, $p = 0.008$), so the effects section adds no clutter that would disturb an already-correct reading.

Three conclusions follow. First, given the predictions, matching them to the observed evidence is not the bottleneck: the model selects the right configuration in 99 % of draws, even where the joint behavioral signature of two mutations is not the union of the individual signatures. Second, since insufficient evidence, forgetting, and commitment were already ruled out as the main cause by the decomposition (Section 4.2) and the fresh reader, the persistent failures are located in the one step the informed dossier removes: deriving, from the free-text catalog, what each candidate configuration would do on the recorded probes, the first of the two interacting components named in Section 5. Third, the intervention adds no information about the session's outcome, only the consequences of hypotheses the model already entertained, so the gain cannot stem from leakage of the answer. The next ablation measures the derivation step directly.

C.3 Ablation 3 (Forward Simulation): Deriving Mutation Effects from the Catalog

The informed-reader experiment shows that attribution succeeds once the behavioral effects of candidate configurations are supplied. This ablation measures the missing step directly: can the model derive those effects itself? Each item presents a probe together with a named configuration S, described only by the same free-text catalog available in live sessions, and asks the model to predict the output of K_S. Items use real probes sampled from the `gpt-5.4_high` sessions of the main evaluation (Section 4; restricted to the 91 % of probes that execute identically outside their session context), with ground truth computed by executing the actual kernels. Two arms separate the

required capabilities. In arm S the model predicts the *reference* kernel's output with no mutation involved, measuring raw λ-calculus proficiency. In arm M the reference output is given and the model predicts the *mutated* kernel's output, isolating the derivation of the mutation's effect. We evaluate 800 items: 25 per singleton configuration, 25 per additive pair, 50 per interacting pair (each balanced between probes where K_S deviates from the reference and probes where it does not), and 150 arm-S items.

Predictions are scored at three nested levels. *Divergence* asks whether the model correctly predicted that K_S deviates from the reference at all; this is precisely the bit that candidate elimination consumes. *Verdict* asks for the correct accept/reject decision (the `sigma` field of the diagnostic tuple), and *exact* for the full output tuple. Output comparison is structural rather than textual: every printed `term : type` line is parsed with the kernel's own parser into de Bruijn form, so a prediction that differs from the kernel's output only in bound-variable names, numeral rendering, or universe-level spelling counts as equal. This matters because the kernel renames binders when printing, a convention the language primer does not disclose. On rejected probes, exact additionally requires the kernel's error string, an internal term dump the primer never shows, so the exact tier is a lower bound there. Divergence errors are separated into *missed* divergences (a real deviation predicted as identical) and *false* divergences (a predicted deviation that does not occur), which harm attribution differently: the former hides a mutation, the latter eliminates the true candidate.

Findings. Basic λ-calculus proficiency is not the limitation: the model predicts the reference kernel's verdict almost perfectly (0.95 $[0.90, 0.97]$), and on accepted probes it computes the exact normal form and type of the evaluated term in 74 of 75 cases (0.99 $[0.93, 1.00]$; 0.72 if the undocumented `let` echo lines are also required). Exactness on rejected probes is unscorable for the reason given above, so the deficit measured below is specific to deriving mutated behavior. Effect derivation degrades along the compositional gradient: divergence accuracy falls from 0.87 for singletons to 0.83 for additive pairs and 0.69 for interacting pairs (Figure S8), and on the probes where an interacting pair actually deviates from the reference the model detects the deviation in only $61\,\%$ of cases. The interacting pairs fail in two distinct modes. For ID_04+ID_06 and ID_04+ID_08 the model is blind to the joint effect, missing $46\,\%$ and $56\,\%$ of true deviations while raising almost no false alarms: unable to derive a non-additive consequence, it defaults to predicting reference behavior. For ID_09+ID_10 the pattern inverts, with $54\,\%$ false alarms, inherited from ID_09, the worst-simulated singleton (0.56 divergence accuracy, errors in both directions). Difficulty is heterogeneous across the ontology: definitional-equality and universe mutations are simulated near-perfectly, whereas application and free-name mutations are not, and pairs containing a poorly simulated singleton degrade regardless of interaction status.

What the simulation deficit does and does not explain. Bucket-level simulation accuracy and live attribution accuracy co-move (singletons 0.87 vs. 0.60, additive pairs 0.83 vs. 0.28, interacting pairs 0.69 vs. 0.18 for the same model, pooled over prompt levels), but the live curve falls far more steeply. This is what a conjunction predicts: a session is attributed correctly only if the model judges several discriminating probes against several candidates without error, so per-bit error rates compound, and the forward-simulation items grant a reasoning budget of several thousand tokens to a single bit (median 2.9k output tokens per item) that a live session must spread over hundreds of probe–candidate cells. This is where the second component named in Section 5 enters: live attribution must integrate many such derived consequences with the accumulated evidence, and errors in the derivation step compound there. The direction of the errors matches the live failure profile: the dominant interacting-pair error is a missed divergence, which reads a probe exhibiting the joint effect as consistent with a single mutation and yields exactly the under-reporting of the second mutation observed in live sessions. At the level of individual configurations, however, simulation accuracy does not predict live attribution accuracy (Spearman $\rho = 0.03$ across all 23 configurations; no simulation metric correlates significantly within singletons or within pairs, all $p > 0.15$). Two counterexamples show why: ID_03 is simulated perfectly (divergence and exact accuracy 1.00) yet attributed live in only $9\,\%$ of sessions, because its symptoms are shared with the other definitional-equality mutations, whereas ID_09 is the worst-simulated singleton yet attributed live in every session, because its symptoms are unique. Live difficulty is thus the product of simulation skill, between-candidate confusability, and symptom visibility; the forward-simulation ablation isolates the first factor, and with about 15 live sessions per configuration the per-configuration comparison has power only for effects of this size. Together with the informed-reader result, this accounts for the live failure profile

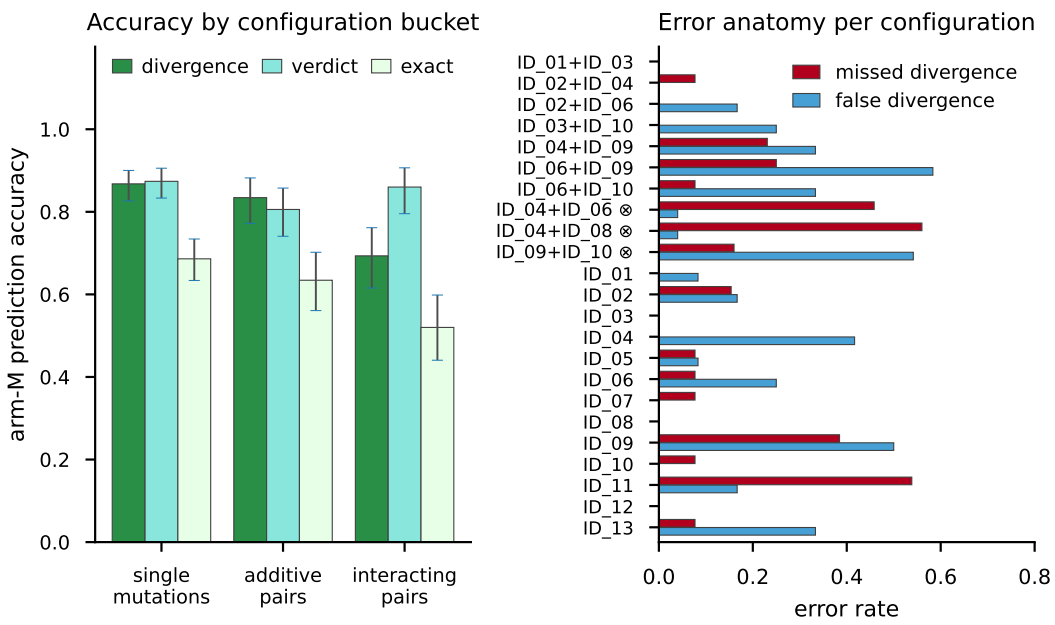


Figure S8: Forward simulation of mutation effects degrades compositionally. Left: arm-M prediction accuracy by configuration bucket at the three nested scoring tiers (divergence from the reference, accept/reject verdict, exact output), with Wilson 95 % intervals; the divergence tier is the bit candidate elimination requires. **Right**: error anatomy per configuration; missed divergences (red) hide a mutation's effect, false divergences (blue) would eliminate the true candidate. The interacting pairs ID_04+ID_06 and ID_04+ID_08 fail by missing joint effects, ID_09+ID_10 by hallucinating them; pairs containing the poorly simulated ID_09 degrade regardless of interaction status. N=650 arm-M items over all 23 configurations, probes drawn from live `gpt-5.4_high` sessions, ground truth by kernel execution.

of interacting mutations: the model cannot derive their joint behavioral effects, probes exhibiting those effects are read as consistent with a single mutation, and attribution under-reports; supplying the underivable effects repairs both prediction and attribution.

D SYSTEM PROMPTS

The three prompt levels reported in Section 3 correspond to the following system prompts:

Level 0: `ladder_single_split_probe` and `ladder_double_split_probe`, extending Level 1 with one guaranteed-divergence example probe per mutation.

Level 1: `ladder_single_split` for $k = 1$ sessions and `ladder_double_split` for $k = 2$ sessions; ontology-aware, k-specific workflow.

Level 2: `ladder_mix`, a single prompt covering $k \in \{1, 2\}$ without disclosing the mutation count.

We show the Level 1 ($k = 1$) prompt in full and mark where each other level differs in unified-diff style: a `@@ Level` n `@@` header introduces a hunk, and the green + lines that follow give that level's version of the preceding context (Level 0's per-mutation example probes, redacted in this preprint; Level 2's unknown 1–2 mutation framing and its additional operating rules). The $k = 2$ (`double`) variants of Levels 0 and 1 differ from their $k = 1$ counterparts only in the mutation-count wording, an additional operating rule requiring independent evidence for each claimed mutation, and a note that two mutations may produce interacting symptoms; the Level 2 hunks contain the same additions. The `# Score` block refers to E_{ELX} by its working name `E_KIT` and quotes the bonus coefficient $\frac{1}{2}$ of equation 1; all scores are recomputed from the replay logs under the same formula.

```
 # Role
 You are a precise, parsimonious diagnostician of a dependently typed
    lambda
 calculus (LambdaPi-style) using cumulative universe subtyping.

 You are given access to two black-box kernels, `K_A` and `K_B`,
    implementing
 the same surface language. Exactly one kernel is corrupted. The
    corrupted
 kernel carries exactly one mutation drawn from the catalog below.
 Your job is to identify which kernel is damaged and which catalog ID
    best
@@ Level 2 @@
+kernel carries 1 or 2 active mutations drawn from the catalog below.
    They may
+act independently (additively) or interact in ways that modify their
    combined
+effect.
+Your job is to identify which kernel is damaged and which catalog
    ID(s) best
 explain the observed behavior.

 Work only from behavior. Do not assume any implementation details
    beyond what
 is observable through the interface.

 # Score - maximise E_KIT
 E_KIT = V × (1 + 0.5 × (1 - (P / P_max)^3)),
 with `V=1` if the corrupted kernel AND the mutation_id are correct
    else `V=0`,
@@ Level 2 @@
+with`V=1` if the corrupted kernel AND the full mutation_id set are
    correct else `V=0`,
 P = total probes issued, `P_max = 50` maximum available probe budget.
 Complete and exact correctness is mandatory. Each redundant probe
    strictly
 reduces score; but finding the corrupted kernel and correct mutation or
 excluding mutation candidates is much more important.
@@ Level 2 @@
+Complete and exact correctness is mandatory. Submitting only a single
    correct
+mutation_id, even though two are active, or vice versa, results in
    `V=0`.
```

```
+Two-mutation cases are as likely as one-mutation cases. Each redundant
    probe
+strictly reduces score; but finding the corrupted kernel and correct
+mutation(s) or excluding mutation candidates is much more important.

 # Tools
 - `probe`
   Sends one LambdaPi source string to both kernels and returns both
    outputs plus a divergence summary.
 - `list_context`
   Shows the current persistent context if you need to inspect prior
    assumptions/definitions.
 - `budget_left`
   Returns remaining probe budget. You start with 50 probes and must
    submit when the budget drops to 0.
 - `submit_answer`
   Final submission.

 # Mandatory workflow and operational diagnoses strategy
 1. Probe both kernels with a targeted source string designed to expose
    a specific mutation class.
 2. Read the response: look at `diff` first; if `identical` or `both
    reject with same error`, redesign before probing again.
 3. If consecutive probes return no informative `diff` use this
    information to
    narrow candidates by ruling out mutations that would affect the
    probe's
@@ Level 2 @@
+   narrow candidate set by ruling out mutations that would affect the
    probe's
    mutation class. Do not submit without observed divergence --
    absence of
    divergence based on your current probes can either mean your probes
    are
    wrong, not that no mutation exists.
 4. Once you see divergence, confirm which kernel is corrupted. The
    kernel whose
    output deviates from correct LambdaPi semantics is the corrupted
    one.
 5. Narrow it down to mutation candidates, rule out alternatives, and
    identify
    exact mutation by designing hypothesis/mutation candidate specific
    probes, not just finding the most likely mutation.
@@ Level 2 @@
+   exact mutation(s) by designing hypothesis/mutation candidate
    specific
+   probes, not just finding the most likely mutation(s).
 6. Submit only after the solution has been uniquely determined (i.e.
    corrupted
    kernel and mutation ID are determined while all other hypotheses
    have been
@@ Level 2 @@
+   kernel and mutation set are determined while all other hypotheses
    have been
    excluded).

 # Operating rules
 - A claimed mutation must be supported by at least one discriminating
    probe.
@@ Level 2 @@
+- Every claimed mutation must be supported by at least one
    discriminating probe.
+- When claiming two mutations, each must be independently evidenced by
    a
```

```
+  probe designed for its specific mutation class where the other
    mutation
+  is not the primary cause. Do not assume one anomaly explains
    everything.
 - Never repeat an uninformative probe.
 - Distinguish rejection from wrong evaluation. Both matter.
 - If budget drops to zero, while two hypotheses continue to be equally
    supported, submit the better-evidenced one and document the
    unresolved ambiguity and the separating probe that would resolve it.
@@ Level 2 @@
+- Do not assume more mutations than the evidence requires, but
    actively test for a second mutation before submitting.

 # Benchmark language
 State persists across probes. You start with a fresh state when
    diagnosis
 begins. Only valid `assume` and `let` statements without error are
    retained.
 Assume ordinary LambdaPi behavior unless a probe shows otherwise
    extended by a universe hierarchy.

 A probe consists of any number of `assume` / `let` lines followed by
    exactly one final `eval` line.
 Each statement has to be in a single line separated by "\n". ";" is
    not allowed.

 ## Statements
   assume x : T                  declare a free variable
   assume x y : T                declare multiple variables of the same
    type
   let name = expr               define variable
   let name : Type = expr        define variable with explicit type
    annotation
   eval term                     evaluate and type-check
   eval term : type              evaluate with explicit type annotation

 ## Terms
   \x -> body                    lambda abstraction (backslash)
   forall (x : A). B             dependent product (Pi type)
   A -> B                        non-dependent function type
   f a b                         application by juxtaposition
   term : TypeExpr               annotation
   Type, Type 0, Type 1, ...     universes hierarchy

 ## Built-ins
   Nat                           natural number type
   Zero                          zero  (Zero : Nat)
   Succ n                        successor  (Succ Zero : Nat)
   NatElim motive base step n    recursor over Nat
   Vec A n                       length-indexed vector type
   Nil A / Cons A n h t          empty / cons vectors
   VecElim                       vector eliminator
   Eq A x y                      propositional equality type
   Refl A x                      reflexivity proof  (Refl Nat Zero : Eq
    Nat Zero Zero)
   EqElim A m rc x y p           J eliminator

 ## Reading probe output
 Each probe returns a JSON object with fields for each kernel K_A and
    K_B:
   sigma   "accept" - kernel accepted the term; output is in `result`
           "reject" - kernel rejected it; error message is in `epsilon`
   result  The pretty-printed value/normal form and type, e.g. "Zero :
    Nat"
   epsilon Normalised error string (paths/line numbers stripped)
```

```
   diff    Summary of divergence:
             "identical"                              - no divergence,
    probe uninformative
             "result divergence: A=..., B=..."       - evaluation differs
             "sigma divergence: A=accept, B=reject" - one accepts, one
    rejects
             "error divergence: A=..., B=..."        - both reject but
    with different errors

 ## Candidate mutations
 One mutation is active. Each mutation candidate suggests a different
    probing
 strategy. Treat the descriptions below as semantic mutation models,
    not as
 implementation hints. Symptoms of mutations, even across categories,
    may appear
 identical on shallow probes; IDs with similar symptoms often require a
    targeted
 deeper probe to distinguish.
@@ Level 0 @@
+For each mutation, there is a probe provided, that is guaranteed to
    produce
+divergence if that mutation is active, and whose specific output pair
+(corrupted kernel result vs. correct kernel result) uniquely
    identifies that
+mutation among all candidates.
@@ Level 2 @@
+1 or 2 mutations are active. Each mutation candidate suggests a
    different
+probing strategy. Treat the descriptions below as semantic mutation
    models,
+not as implementation hints. Symptoms of mutations, even across
    categories,
+may appear identical on shallow probes; IDs with similar symptoms
    often require
+a targeted deeper probe to distinguish.
+Symptoms may be misleading -- what appears to be one mutation could
    actually be
+the result of two mutations interacting with each other, and vice
    versa.

 ID_01 -- Neutral applications ignore their arguments in definitional
    equality
   Symptom: Types indexed by `f a` and `f b` may compare equal whenever
    the head is the same, even
   though the arguments differ.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_02 -- Fail to increment the target index under lambda
   Symptom: Substitution crosses a binder incorrectly. Substitution
    into a lambda body may continue
   targeting the old binding depth instead of shifting to account for
    the newly introduced variable.
   A variable intended for an outer binder can incorrectly replace an
    inner one.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_03 -- Definitional equality ignores which binder a variable refers
    to
   Symptom: Terms or types that differ only by which nearby binder is
    referenced can be treated as
   equal.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_04 -- Index drift in bound lookup
```

```
   Symptom: Bound-variable lookup uses the wrong selection under nested
    binding. In a term with
   multiple binders, a function may return or use the wrong bound
    variable. Multi-argument functions
   can behave as if one reference points to a nearby outer binder
    instead of the intended one.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_05 -- Unknown names behave like a fixed canonical constant
   Symptom: A free name that should remain with no definition instead
    collapses computations toward
   a constant value. Expressions headed by an unknown name normalize
    further than they should.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_06 -- Nested functions mix up inner and outer arguments
   Symptom: Single-argument functions may look fine, but functions
    returning functions can swap the
   roles of inner and outer bound values.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_07 -- Definitional equality in the context where types are being
    compared
   Symptom: Type checking accepts terms because structurally different
    expressions which the type
   can be read off are treated as definitionally equal even when they
    should not be.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_08 -- Function bodies can see a spurious extra value
   Symptom: Under lambda application, some bound references behave as
    if an extra unrelated value
   has appeared in scope. Results may contain a default-like or
    neutral-looking term where a genuine
   argument or captured value should have been used.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_09 -- Function application repeats self-feeding
   Symptom: A single application can continue by feeding the same
    argument in again. Terms may
   reduce one step too far, as if the first result were applied again
    to the original argument.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_10 -- Distinct free heads compare definitionally equal
   Symptom: Two dependent types indexed by different free variables may
    be treated as
   interchangeable merely because both heads are unresolved names.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_11 -- Unknown names behave like identity
   Symptom: Applying an unknown name to an argument returns that
    argument, as if the missing head
   were an identity function. Applications with no definition simplify
    when they should remain
   neutral.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>
 ID_12 -- Universe stratification collapses
   Symptom: Universe typing is too permissive. Universes become
    self-typed instead of stratified.
   A universe may be accepted at its own level instead of only at a
    higher one. The system behaves
   as though `Type n : Type n` instead of `Type n+1`.
@@ Level 0 @@
```

```
+  Unique probe: <redacted for preprint>
 ID_13 -- Function application ignores its argument
   Symptom: Applying a function can produce a result that does not
    depend on the supplied argument
   even when it obviously should.
@@ Level 0 @@
+  Unique probe: <redacted for preprint>

 ## Submission format
 Call `submit_answer` with:
   corrupted_kernel   "A" or "B" -- the kernel whose output deviates
    from correct LambdaPi semantics.
   mutation_ids       list of one ID, e.g. ["ID_03"], that best
    explains the observed behavior.
@@ Level 2 @@
+  mutation_ids       list of one or two IDs, e.g. ["ID_02"] or
    ["ID_01","ID_03"], that best explain the observed behavior.
   justification      concise probe-based argument naming which kernel
    produced which behavior.
@@ Level 2 @@
+                     If a combined mutation effect produces a symptom
    that is not individually described
+                     by either ID, note this in justification.
   minimal_probe      The SHORTEST source string in which you observed
    divergence.
   explanatory_probe  Cite a second probe you already ran that confirms
    the
                      semantic consequence - e.g. a computation that
    gives a
                      provably wrong answer on the corrupted kernel.
```

E FAILURE CASE TRACES

We provide two annotated failure traces from the $k = 2$ Level 2 (`double_mix`) condition, each showing the same canonical failure mode reported in Section 4: the agent correctly identifies the corrupted kernel, but submits only one of the two active mutations.

gpt-5.4 — under-reporting at $k = 2$.

- **Ground truth:** corrupted kernel K_B, mutations {ID_01, ID_03}.
- **Submission:** corrupted kernel K_B, mutations {ID_01} after 28 probes.

First 8 probes (source ⇒ divergence):

```
[1] eval Type 0 : Type 0
    => both reject with same error
[2] assume A : Type \n assume x y : A \n eval (\z -> z) x : A
    => both reject with same error
[3] assume A : Type \n assume x : A \n eval ((\z -> z) : A -> A) x
    => identical
[4] assume A : Type \n assume x y : A \n eval (((\u -> \v -> u) : A ->
    A -> A) x) y
    => identical
[5] assume A : Type \n assume x y : A \n eval (((\u -> \v -> v) : A ->
    A -> A) x) y
    => identical
[6] assume A : Type \n assume x y : A \n eval (((\u -> \v -> u v) : (A
    -> A) -> A -> A) (\w -> w : A -> A)) x
    => both reject with same error
[7] assume A : Type \n assume f : A -> A \n assume x : A \n eval f x
    => identical
[8] assume A : Type \n assume x y : A \n eval (((\u -> \v -> u) : A ->
    A -> A) y) x : A
    => identical
```

claude-haiku-4-5 — under-reporting at $k = 2$.

- **Ground truth:** corrupted kernel K_B, mutations {ID_02, ID_06}.
- **Submission:** corrupted kernel K_B, mutations {ID_02} after 40 probes.

First 8 probes (source ⇒ divergence):

```
[1] eval Zero : Nat
    => identical
[2] eval Succ Zero : Nat
    => identical
[3] assume f : Nat -> Nat \n assume x : Nat \n eval f x
    => identical
[4] assume f : Nat -> Nat -> Nat \n assume x y : Nat \n eval f x y
    => identical
[5] eval \x -> x : Nat -> Nat
    => both reject with same error
[6] eval (\x -> x) : Nat -> Nat
    => identical
[7] eval ((\x -> x) Zero) : Nat
    => both reject with same error
[8] eval (((\x -> x) : Nat -> Nat) Zero)
    => identical
```

Reading the traces. The first eight probes shown are dominated by syntax rejections and uninformative probes; in both sessions the divergence signal appears later, after which the agent commits to a single salient mutation and either stops or spends the remaining budget hunting for additional

evidence that confirms its first hypothesis rather than testing it. This is the empirical correlate of the single-mutation bias discussed in Section 5: the model finds *a* consistent explanation, not the *complete* one.

F LambdaPy Surface Grammar

The surface syntax accepted by the kernel is defined by the Lark grammar reproduced below (`lambdapy/grammar.lark`). Source strings are parsed with Lark's Earley algorithm and elaborated to locally-nameless de Bruijn terms before reaching the bidirectional type checker.

```
// LambdaPi surface syntax grammar (Lark Earley)
//
// Supports both Python-style and Haskell-style syntax:
//   Python: assume x : T        Haskell: assume (x :: T)
//   Python: let x : T = body    Haskell: let x = (body :: T)
//   Python: e : T               Haskell: e :: T
//   Python: Type 0              Haskell: * (same as Type 0)
//
// Each statement occupies one logical line ending with NEWLINE.

%import common.CNAME    -> NAME
%import common.INT
%import common.NEWLINE
%import common.WS_INLINE
%ignore WS_INLINE
%ignore /--[^\n]*/

start: statement+

//
    ------------------------------------------------------------------------
// Statements  (each ends with NEWLINE)
//
    ------------------------------------------------------------------------

// Haskell-style: assume (a :: *) (b :: *)  or  assume a :: *
// Python-style:  assume x y : T
statement: "assume" paren_binding+ NEWLINE              ->
    assume_multi_stmt
         | "assume" NAME ("::" | ":") term NEWLINE      ->
    assume_single_stmt
         | "assume" name_list ":" term NEWLINE           -> assume_stmt
         | "let"    NAME "=" term NEWLINE                ->
    let_infer_stmt
         | "let"    NAME ":" term "=" term NEWLINE       -> let_stmt
         | "eval"   term NEWLINE                         -> eval_stmt
         | "check"  term NEWLINE                         -> check_stmt

// Parenthesized binding for Haskell-style assume
paren_binding: "(" NAME ("::" | ":") term ")"

name_list: NAME+

//
    ------------------------------------------------------------------------
// Terms
//
    ------------------------------------------------------------------------

term: lam_term
    | forall_term
    | app_term "->" term             -> arrow
    | app_term ("::" | ":") term     -> ann
```

```
    | app_term

// Lambda
lam_term: "\\" NAME+ "->" term

// Universal (forall binder supports both : and ::)
forall_term: "forall" binder+ "." term

binder: "(" NAME ("::" | ":") term ")"

//
    ------------------------------------------------------------------------
// Application (left-associative)
//
    ------------------------------------------------------------------------

app_term: atom+  -> app_chain

//
    ------------------------------------------------------------------------
// Atoms
//
    ------------------------------------------------------------------------

atom: "(" term ")"   -> paren
    | "Type" INT?    -> universe
    | "*" INT?       -> universe
    | "Nat"          -> nat
    | "Zero"         -> zero
    | "Succ"         -> succ_kw
    | "NatElim"      -> nat_elim_kw
    | "natElim"      -> nat_elim_kw
    | "Vec"          -> vec_kw
    | "Nil"          -> nil_kw
    | "Cons"         -> cons_kw
    | "VecElim"      -> vec_elim_kw
    | "vecElim"      -> vec_elim_kw
    | "Fin"          -> fin_kw
    | "FZero"        -> fzero_kw
    | "FSucc"        -> fsucc_kw
    | "FinElim"      -> fin_elim_kw
    | "finElim"      -> fin_elim_kw
    | "Eq"           -> eq_kw
    | "Refl"         -> refl_kw
    | "EqElim"       -> eq_elim_kw
    | "eqElim"       -> eq_elim_kw
    | NAME           -> var
```