LOGIC-AWARE POLICY HALLUCINATION DETECTION THROUGH TEXT-TO-GRAPH ENCODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Translating policy text into an executable program can omit conditions needed for the correct decision. As a result, policies that should produce different decisions for the same question and context may produce identical outcomes after translation. This loss of semantic sensitivity contributes to hallucination when the system asserts an unsupported conclusion. Successful symbolic execution alone cannot establish source support, so automatic detection must connect source meaning with the completed claim. We learn continuous, query-conditioned graph embeddings to recognize such failures in supplied verifier runs. A graph-only branch predicts candidate execution, while a source-aware branch aligns policy units with program nodes and scores each asserted claim without requiring a comparison partner. A paired branch uses symmetric and directional features to classify changes in semantic sensitivity. Logic-derived supervision trains these differentiable representations without sending gradients through the compiler or symbolic evaluator. Source-grounded repair and replay assess contribution separately. On 10,845 held-out controlled pairs from 600 policy families, sensitivity accuracy is 63.89%. Individual-claim detection achieves 81.24% F1 and 91.88% average precision (AP) on 4,635 evaluable assertions; the constant-score AP baseline is 65.46%. On a structurally broader benchmark, AP is 56.56% among scored claims; a subsequent capacity check scores all 3,974 evaluable claims and yields 56.44% AP. These single-seed results establish a detection signal in controlled logical settings alongside substantial transfer failures. Outcome recoverability, executable fidelity, and source-supported claims remain distinct.

1 INTRODUCTION

An answer may cite the correct policy and still assert an unsupported permission. Consider a policy whose only authorization rule permits employee access with manager approval. If Policy Verifier drops the approval requirement when translating the policy into an executable program, it can prove access for an employee without approval. The proof follows the generated program, but the resulting answer goes beyond what the policy supports.

The missing requirement removes a distinction that should affect the decision: approved and unapproved employees now receive the same permission. We call this a loss of *semantic sensitivity*. The program's *answer region*—the set of cases receiving permission—has expanded to include cases the policy does not authorize. This expansion leads to hallucination when the answer asserts the unsupported permission. A defect may both admit unauthorized cases and exclude cases the policy permits.

Automatic detection must relate the original policy to the generated program and the answer actually given. A structural difference alone is insufficient evidence of an error: equivalent rewrites may preserve every decision. The challenge is to make this comparison across unfamiliar logical structures without a trusted reference program at inference.

These comparisons support three distinct tasks. *Detection* asks whether the policy supports the answer as asserted. *Localization* identifies a defective artifact or processing step. *Attribution* tests whether correcting that defect changes the claim or its justification. In the approval example, we restore the missing condition and repeat the later steps with the same question and context. We check separately whether the program still permits access and whether the answer still asserts permission.

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.

Our detector scores individual claims and compares pairs of runs. To assess one claim, it aligns policy text with program nodes and combines this alignment with the question, context, and recorded verifier steps. Paired analysis compares two supplied runs for the same question and context, asking whether their programs preserve the effect of a policy change. Comparison features capture both shared content and the direction of change. Neither run is a trusted reference, and scoring one claim requires no comparison partner.

Graph encoders map discrete programs to continuous vectors, or *embeddings*. A graph-only head predicts the candidate program's outcome; source-aware layers use the representations to assess claims and paired behavior. A supervised loss compares these predictions with logic-derived labels. Backpropagation then computes gradients—how the loss changes with trainable parameters—to guide learning.

The learned representation and the executed decision must still be assessed separately. An embedding may retain enough information to predict the policy's required decision even when its program executes incorrectly. We analyze this *recoverability* using information theory, keeping it distinct from correct execution and support for the answer actually asserted.

We learn differentiable, query-conditioned representations of policy programs and verifier traces to detect decision-relevant discrepancies and unsupported claims. Logic-derived supervision jointly trains graph encoders, source alignment, and prediction heads; supplied programs and symbolic execution remain fixed. Held-out evaluations show a detection signal on the original controlled corpus, with substantial transfer failures on the broader benchmark. Development-selected thresholds produce many false alarms on both test populations. Source-validated repair and replay separately test whether a defect contributed to the claim or its justification.

2 Related work and distinction

FActScore measures source support for atomic claims, while RAGTruth provides annotated examples of unsupported generation (Min et al., 2023; Niu et al., 2024). Policy verification adds an executable intermediate representation: an answer can correctly follow a program that omits a source constraint. Answer agreement cannot establish faithfulness when both programs omit the same constraint. Our detector relates policy conditions to candidate behavior and asserted claims.

FoVer uses formal verification to derive step-level supervision (Kamoi et al., 2025). Policy autoformalization uses a generator, deterministic checks, and an LLM critic to translate instructions into Cedar (Mondl et al., 2026). Our detector learns representations of supplied programs and completed claims. Logic-derived sensitivity and claim-support labels jointly train graph encoders, source alignment, and prediction heads through backpropagation, while program construction and symbolic execution remain fixed.

CheckList's invariance and directional tests connect directly to semantic sensitivity (Ribeiro et al., 2020): equivalent rewrites should preserve decisions, whereas a decision-relevant policy edit should change them appropriately. Graph Matching Networks learn graph similarity (Li et al., 2019). We specialize graph comparison to whether programs preserve the behavioral effect of a policy edit. Shared encoders represent typed dependencies within each supplied candidate run; neither branch is a trusted reference. Source–node cross-attention provides local alignment before pooled comparisons combine source, program, query, and checkpoint features. A separate single-run head assesses each asserted claim.

Semantic entropy measures uncertainty across generated answer meanings (Farquhar et al., 2024). Our analysis asks whether reference outcomes remain recoverable from a graph and query. Recoverability does not ensure that executing the graph produces those outcomes: an incorrect operator can preserve information while reversing behavior. Section 4 develops this distinction through information-theoretic probing.

CausalGaze intervenes on graphs derived from LLM internals (Kong et al., 2026). Here, checkpoint scores identify suspected divergence in explicit verifier artifacts. Source-grounded repair and downstream replay separately test whether a defect contributed to an unsupported claim. This separates neural detection from intervention-supported contribution. Appendix H provides detailed comparisons.

3 SEMANTIC SENSITIVITY AND POLICY SUPPORT

We measure semantic sensitivity by asking whether a policy change that matters to a decision changes program behavior appropriately. This requires a logical reference for what the policy supports. The resulting comparisons supervise a differentiable graph detector: gradients train its representations and predictions, while policy interpretation and symbolic execution remain fixed.

The reference language covers Boolean conditions, permissions, prohibitions, scoped exceptions, acyclic priorities, and obligations with finite deadlines over finite domains (Appendix C). Policy text X admits a finite, nonempty set $\mathcal{P}^*(X)$ of interpretations validated independently of the candidate program. Each controlled policy has one validated interpretation.

The original question q^{txt} has validated meaning q^*, distinct from candidate query $\widehat{q}$. Context C records authenticated evidence, evaluation time, and observation completeness; $Q = (q^*, C)$ fixes the reference question and context. Missing observations admit completions $c \in \Omega(C)$ consistent with that evidence. Each interpretation/completion pair (P, c) defines a reference scenario.

For proposition a, the indicator $s_{P,c}(a)$ records support under that scenario; a separate indicator $r_{P,c}(a)$ records support for its declared normative opposite $\bar{a}$. Permission and prohibition oppose each other for the same action and scope; lack of permission is not prohibition. Together they define

$$Y_{P,c}(a) = (s_{P,c}(a), r_{P,c}(a)) \in \{0,1\}^2. \tag{1}$$

Figure 1 shows the four outcomes. Each requires completed checks: invalid artifacts, unsupported fragments, and failed checks remain unresolved. Pending obligations retain their temporal state.

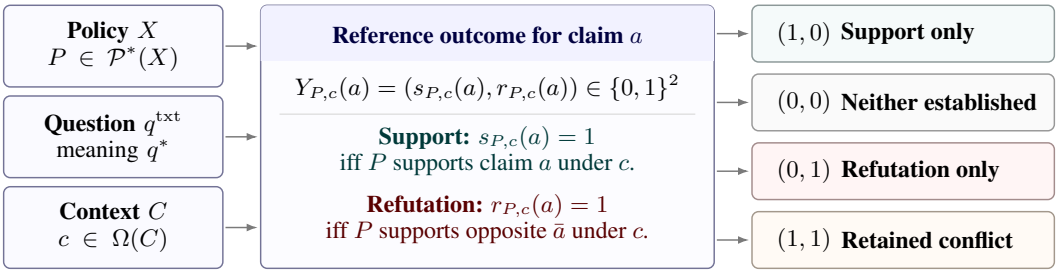


Figure 1: Reference outcomes across policy interpretations and context completions.

Hallucination labels. A completed answer must be assessed by both its claim and how it presents that claim. Assertion mode $u = 1$ presents a as settled; $u = 0$ explicitly acknowledges conflict. Pure abstention asserts no proposition and receives no claim label.

Within one scenario, hallucination means missing support or conflict presented as settled. Writing s, r for its indicators gives $h(s, r, u) = \mathbf{1}[s = 0 \vee (s = r = u = 1)]$, where $\mathbf{1}[\cdot]$ equals one when the condition holds and zero otherwise. The final label $\mathsf{Hall}(a, u) = b$, with $b \in \{0, 1\}$, requires every scenario check to complete and agree:

$$\mathsf{Hall}(a, u) = b \quad \Longleftrightarrow \quad h(s_{P,c}(a), r_{P,c}(a), u) = b \quad \forall (P, c) \in \mathcal{P}^*(X) \times \Omega(C). \tag{2}$$

Disagreement, failed checks, or an ambiguous claim leave the label unresolved. Missing support and suppressed conflict are reported separately. The full query outcome $Y^*(X, Q)$ requires completed checks agreeing on both coordinates for proposition $a(q^*)$, a stronger condition than agreement on its claim label. With X fixed, we write $Y^*(Q)$.

Pipeline checkpoints. Checkpoints record how a claim develops. Event ℓ stores semantic output G_ℓ, source links A_ℓ, candidate query, and execution status. Its recorded predecessors $\mathrm{pa}(\ell)$ supply its inputs. These dependencies form the event DAG $\mathcal{T}$, with repeated calls represented as separate events. This graph connects processing steps, whereas a policy-expression DAG connects logical operators and conditions. Table 1 lists the meaning each stage must preserve.

Table 1: Policy and query preservation requirements. Replacing AND with OR, for example, admits cases satisfying only one condition. Such a defect need not affect the current claim.

Processing stage	**Meaning to preserve**	**Example violation**
Source interpretation	Scope, modality, condition	Approval requirement omitted
Graph construction	Argument roles, dependencies	Exception attached to wrong rule
Compilation	Bounded program behavior	AND replaced by OR
Query formalization	Proposition, bindings, time	Obligation becomes permission
Selection and proof	Applicable rules and opposites	Applicable prohibition excluded
Answer generation	Claim support and assertion mode	Unresolved reported as settled

Execution disagreement. To measure executable fidelity, compare candidate program G with validated reference P^* on cases Q with complete reference outcomes. Write $V(G, Q)$ for the candidate outcome and let μ weight the cases. The disagreement rate $D_\mu(G, P^*) = \mathbb{E}_\mu \mathbf{1}[V(G, Q) \neq Y^*(Q)]$ averages their mismatch indicator. It lies in $[0, 1]$; uniform weights give the fraction of mismatching cases. Failed execution counts as a mismatch and is also reported separately (Appendix C.4).

Answer regions and semantic sensitivity. An *answer region* contains the cases assigned the same outcome. For a fixed reference P^* and bounded domain $\mathcal{D}$ of cases with complete reference outcomes, define

$$\mathcal{R}_G(y) = \{Q \in \mathcal{D} : V(G, Q) = y\}, \qquad \mathcal{R}^*(y) = \{Q \in \mathcal{D} : Y^*(Q) = y\}. \tag{3}$$

Dropping an approval requirement can expand permission to unapproved employees: $\mathcal{R}^*((1, 0)) \subsetneq \mathcal{R}_G((1, 0))$. The program then ignores a condition that should distinguish decisions. Other defects can both admit unsupported cases and exclude supported ones, without either region containing the other (Appendix B.3).

The paired test holds Q fixed and changes policy X to X'. Let $G_{\mathrm{PV}}(X)$ denote its program under fixed verifier settings and replay context. A required distinction is lost when reference outcomes differ but executed outcomes coincide:

$$S_{\mathrm{lost}}(X, X'; Q) = \mathbf{1}[Y^*(X, Q) \neq Y^*(X', Q)]\, \mathbf{1}[V(G_{\mathrm{PV}}(X), Q) = V(G_{\mathrm{PV}}(X'), Q)]. \tag{4}$$

This detects lost changes, not every error. Swapping two distinct reference outcomes gives $S_{\mathrm{lost}} = 0$ despite two wrong decisions. Each execution must therefore also be compared with its reference. Equivalent wording and query-irrelevant edits require unchanged behavior; failed executions leave this diagnostic unresolved.

These discrete comparisons supervise the differentiable detector. A sensitivity failure alone is not hallucination: the answer must assert a claim with missing support or present conflict as settled. Appendix B.4 gives finite examples.

4 Information Preservation and Policy Fidelity

A representation may retain enough information to predict the correct decision even when its program executes incorrectly. We distinguish this *recoverability* from executable fidelity.

Policy and query paths. Policy Verifier jointly constructs program G and query $\widehat{q}$; graph encoders produce embeddings $Z_\phi^{\mathrm{cand}} = E_\phi(G, \widehat{q}, C)$ (Figure 2).

Outcomes are comparable only when their questions match. Permission and obligation can both receive $(1, 0)$ yet express different claims. Descriptors $\tau^*, \widehat{\tau}$ record modality, bindings, scope, and time; δ_{target} flags a mismatch. For aligned, completed checks, δ_{exec} compares Y_{exec} with Y^*. Alignment alone does not prove query equivalence; incomplete comparisons remain unresolved.

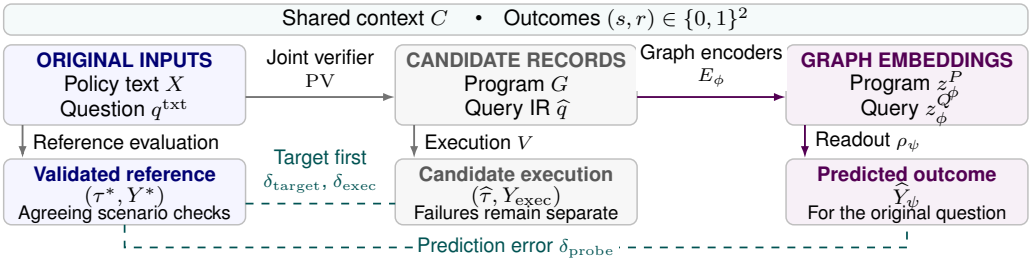


Figure 2: Policy and query paths under shared context C. Solid arrows show computation; dashed links compare results. Check the question's target (τ) before comparing executed outcomes (Y). The readout separately predicts the reference outcome. Reference checks must complete and agree; their outcomes remain evaluation targets. Correct prediction does not establish faithful execution.

Readout ρ_ψ predicts the original question's reference outcome from embeddings; δ_{probe} compares it with Y^*. The routes commute when they agree throughout the declared domain (Appendix A.7).

Information retained by an embedding. Using validated $Q = (q^*, C)$ isolates policy-graph information from candidate-query errors. Its embedding $z_\phi(G, Q)$ differs from Z_ϕ^{cand}. For a common population with complete reference outcomes, conditional entropy H measures remaining uncertainty:

$$E^G = H(Y^* \mid G, Q), \qquad E^z = H(Y^* \mid z_\phi(G, Q), Q). \tag{5}$$

Lower entropy means greater recoverability. With the trained encoder fixed, use deterministic graph/query-only inputs, excluding source text, labels, and provenance. Then $E^z \geq E^G$: encoding cannot add outcome information. Equality preserves it; a positive gap measures loss in bits. Learning can still make retained information easier to predict.

Why recoverability differs from execution. Two equally likely policies with different outcomes but identical (G, Q) leave one bit of uncertainty (Proposition 1). Source-aware monitoring can inspect the differing condition. Conversely, complementing binary outcomes preserves information but reverses execution: $V(G, Q) = 1 - Y^*$ gives $E^G = 0$ and $D_\mu = 1$ (Proposition 2). Hallucination still requires a claim with missing support or conflict presented as settled (Appendix B).

Assessing recoverability with probes. A *probe* predicts reference outcomes from a graph or embedding and Q, with the encoder fixed and disjoint fitting/evaluation families. Cross-entropy penalizes low probability for the correct outcome. High loss can reflect missing information or predictor error; it alone does not establish information loss (Appendix A.5; Pimentel et al., 2020; Voita & Titov, 2020).

The detector's execution head predicts candidate behavior. A diagnostic probe instead targets reference outcomes and detaches its embedding, so its loss updates only the readout. No independent probing, entropy reduction, or information-preservation training benefit is reported.

5 Encoding Programs, Sources, and Verifier Traces

The detector combines executable structure with source requirements to assess a completed claim. Shared encoders support candidate-outcome prediction, individual-claim detection, and paired sensitivity analysis (Figure 3).

Inputs and prediction tasks. A run records policy text, the original question and context, candidate program and query, proof and answer, assertion status, and checkpoints. Individual-claim detection uses one run. Sensitivity analysis needs two supplied runs L, R: holding question and context fixed lets us compare a policy change with its effect on executed behavior. Neither run is a trusted reference. Independently validated outcomes provide training labels, never inference inputs.

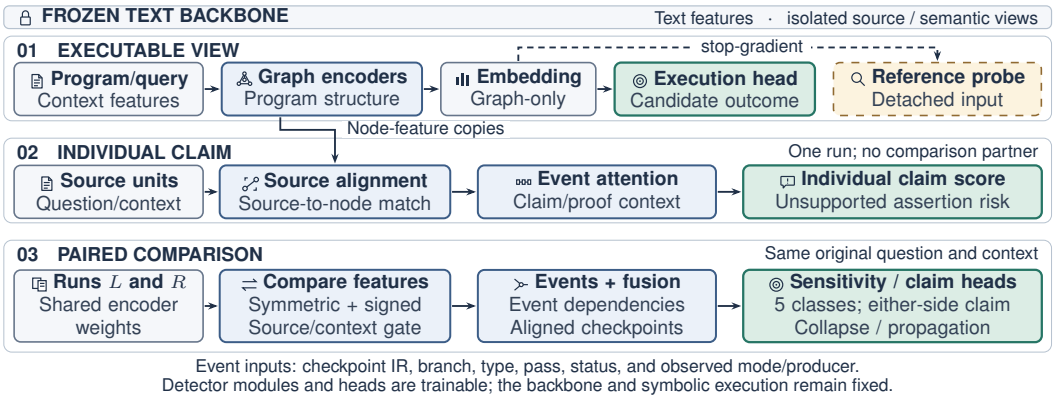


Figure 3: Shared encoders support three inference paths. Source alignment uses node-feature copies while the graph embedding remains source-free. The dashed probe path stops gradients. Checkpoint scores indicate suspected divergence; contribution requires separate repair and replay.

Representing text and executable structure. A frozen language model supplies token features (IBM Granite Team, 2025); trainable projections represent source, question, context, and answer/proof. Graph encoders represent the program and intermediate representations (IRs), including the candidate query. Typed relations and structural attention encode operators and dependencies; local messages exchange information between connected nodes. Query-conditioned pooling combines node features into one vector per graph. Corresponding encoders share weights across runs. Graph-only inputs include executable records and query/context, excluding source text, provenance, construction modes, and labels. Appendix D specifies dimensions and budgets.

Aligning source requirements with program nodes. A program can execute successfully while omitting a source requirement. To expose such discrepancies, bidirectional attention relates source units to copies of program-node features, conditioned on question and context. Source units are bounded token groups, not validated clauses. A learned null node provides an unmatched option, so a source unit need not select an existing program node. Pooling these comparisons gives an alignment summary without overwriting the source-free graph embedding. Attention weights provide learned correspondence features; they do not establish that a constraint is missing.

Following changes through checkpoints. Checkpoint features describe how artifacts evolve along a run. Each event combines IR features with branch, type, compiler pass, execution status, and recorded construction mode and producer. Recorded correspondences link events to aligned source features. Dependency attention connects events through producer–consumer edges, which differ from edges within a program graph. The individual path processes one run, conditioned on question, context, and answer/proof. The paired path uses separate attention blocks and matches events by boundary and invocation number, preserving repeated calls and unmatched observations. Its aligned-event states are denoted h_j.

Comparing the two runs. The paired path asks what changed and in which direction. For each representation type m—source, question, answer/proof, program, or query—let $v_L^m, v_R^m \in \mathbb{R}^{256}$ denote the two vectors. Their comparison is

$$d_m = P_m([v_L^m + v_R^m;\ |v_L^m - v_R^m|;\ v_L^m \odot v_R^m;\ v_R^m - v_L^m]), \tag{6}$$

where $[\,;\,]$ concatenates features and P_m is a learned projection. Sum, absolute difference, and elementwise product ($\odot$) capture shared content, change magnitude, and interactions; they are unchanged when the runs are exchanged. The signed difference retains the direction from L to R. A source/context comparison separately predicts whether the change matters to the decision. Supervision connects these features to behavior; vector distance alone does not establish a defect.

Predicting claim risk and sensitivity. The graph-only execution head predicts the candidate's outcome. For individual-claim detection, claim-conditioned attention pools one run's events and

combines them with source, alignment, question, context, and answer features. The resulting score assesses a completed assertion; nonasserting or failed runs remain unscored.

The paired predictor instead combines event and comparison features:

$$g = F([\bar{h} + d_A; d_{\text{program}}; d_{\text{query}}; d_{\text{source}}; d_{\text{question}}; d_{\text{answer}}; d_\Delta]),$$

where F is a learned fusion network, $\bar{h}$ averages paired-event states, d_A compares alignment summaries, and d_Δ summarizes change statistics. Heads predict one of five sensitivity classes and whether either run contains an unsupported claim. Auxiliary targets cover relevance, candidate change, and ordered-pair error. Paired checkpoint heads use constructed collapse and propagation labels. Individual localization and contribution losses are masked because their labels are unavailable. Checkpoint scores indicate suspicion; attribution requires the interventions in Section 6.

Learning the detector. Logic-derived labels supervise candidate outcomes, sensitivity classes, and claim support. Differentiability supplies gradients for updating neural parameters. For pair i and loss term k, let $\mathcal{O}_{ik}$ contain n_{ik} available labels, with per-label loss ℓ_{ikr} and weight λ_k. A batch of N pairs uses

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \sum_{k:n_{ik}>0} \frac{\lambda_k}{n_{ik}} \sum_{r \in \mathcal{O}_{ik}} \ell_{ikr}. \tag{7}$$

Each term averages losses over its available labels; weighted terms are summed and averaged across pairs. Active weights are one; unavailable labels contribute no loss. Appendix D specifies the losses. Gradients jointly update graph encoders, source alignment, event and comparison layers, and prediction heads. The pretrained backbone, compiler, candidate programs, symbolic execution, and labels remain fixed. The reference-outcome probe receives a detached embedding: its loss updates only the probe and supplies no information-preservation training term.

6 From discrepancy witnesses to claim attribution

A detector score prioritizes a claim for inspection. Attribution requires evidence linking a defect to that claim. We establish a behavioral discrepancy, trace its earliest observed occurrences, and test contribution through validated repair and replay.

Establishing a discrepancy. A *discrepancy witness* is a question and context for which completed candidate and validated reference evaluations disagree. Both must answer the same interpreted question. For permission, let s_G and s^* denote candidate and reference support. Then $s_G = 1, s^* = 0$ identifies permission supported by the program without source authorization, even if no prohibition applies. This establishes a behavioral error; claim emission and the error's origin require separate evidence (Proposition 3, Appendix A).

Finding no witness establishes equivalence only after a completed exhaustive or sound-and-complete check over the declared domain. Ambiguous policies and incomplete observations require agreement across admissible reference scenarios. Temporal comparisons fix evaluation time and observation completeness; unfinished checks remain unresolved.

Tracing the observed discrepancy. Starting from the emitted claim, follow recorded dependencies backward. At each boundary, check the stage's preservation requirement and agreement with the source. Faithful inputs yielding a defective output indicate a newly introduced discrepancy; persistence of the same discrepancy from input to output indicates propagation. The earliest validated occurrences along these paths can form a set across branches. Missing predecessors leave earlier origin unresolved. Track correction as well as propagation: a defect corrected before emission may no longer affect the claim. These observations identify components to test.

Testing contribution by repair and replay. Replace a suspect component, or a declared set of components, with independently source-validated alternatives. Keep the source, original question, context, reference interpretation, and software and configuration fixed, then replay dependent stages. Compare the same original proposition a in the same assertion mode before and after repair.

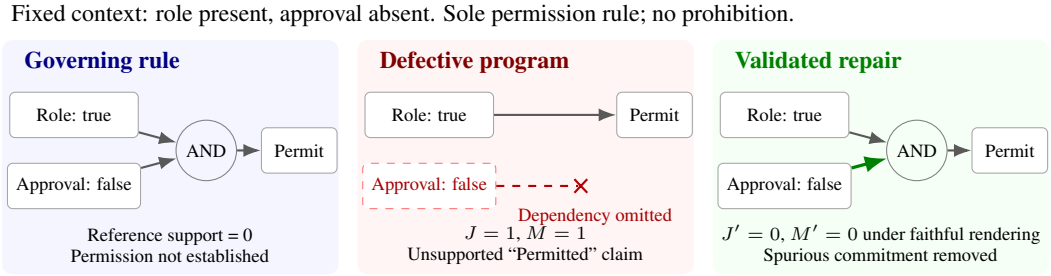
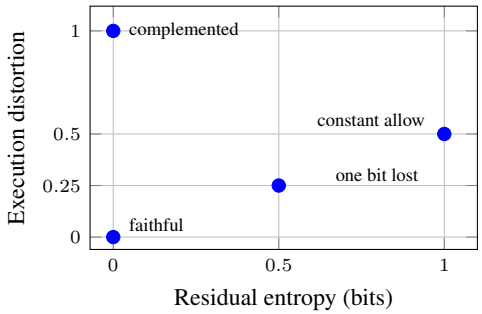


Figure 4: Illustrative repair with fixed source support. Restoring the approval requirement removes candidate proof support and, under the assumed faithful answer renderer, the unsupported permission claim. Absent approval establishes no prohibition in this example. This is an exact construction, not a measured model result.



Encoding	E	$\mathcal{R}_B$	D	D^+
Faithful	0	0	0	0
One bit lost	.5	.25	.25	.25
Constant allow	1	.5	.5	.5
Complemented	0	0	1	.5

$\mathcal{R}_B$: optimal prediction error.
D: literal execution disagreement.
D^+: unsupported-permission mass.
All values are exact constructions.

Figure 5: Exact constructions separate recoverable information, execution disagreement, and unsupported-permission mass. Interpreting the latter as unsupported claim emission additionally assumes a faithful answer renderer. Appendix B gives encodings and derivations.

Repair can affect the proof and the answer differently. For run π, $J_\pi(a)$ records completed candidate proof support for a; $M_\pi(a)$ records whether the answer asserts a in the original mode. Incomplete binding or proof checks leave J unresolved. Refutation remains separate: support amid conflict cannot justify a settled assertion. Source support stays fixed throughout repair (Figure 4).

Contribution requires correction of the intended discrepancy and an effect on justification or emission. If repair removes proof support while the answer still asserts a, the unsupported claim persists. Report both effects separately, and check any newly emitted claims against the source.

The deterministic evaluation tests all subsets of compiled-program, query, and answer repairs, with null and reverse controls (Appendix E). Findings are conditional on these replacements and fixed inputs; they establish neither a unique cause nor hidden neural reasoning.

7 Analytical Examples and Controlled Evaluation

Exact examples separate recoverability from correct execution. We then evaluate claim detection and paired sensitivity. Score coverage and unresolved cases remain separate.

Analytical controls. A uniform reference pair $B = (B_0, B_1) \in \{0,1\}^2$ and an independent uniform query $Q \in \{0,1\}$ define $Y^* = B_Q$: eight equally likely cases. Figure 5 compares residual entropy E with execution disagreement D. Complementing both bits preserves answer information while reversing execution. Unsupported-permission mass D^+ equals unsupported emission only when the renderer asserts permission exactly when execution returns one.

Corpora and evaluation. Both corpora use constructed policies, controlled-language renderings, and deliberately varied candidate artifacts. A *family* groups a reference policy, its renderings, question/context cases, and mutations; related examples stay within one split. Claim labels assess source

Table 2: Claim detection at raw threshold 0.5; metrics and coverage in percent. P/R: precision/recall; AP: average precision; FPR: false-positive rate. Coverage is scored/evaluable claims. Standalone detection metrics use scored claims; development has reused structures and prior selection exposure.

Population	Scored/evaluable	Coverage	P	R	F1	AP	FPR
Development	1,338/1,338	100.00	99.44	97.78	98.60	99.97	1.15
Original test	4,635/4,635	100.00	84.77	77.98	81.24	91.88	26.55
Standalone test	2,970/3,974	74.74	52.92	26.73	35.52	56.56	46.71

support, not the presence of a defect alone. For example, dropping approval makes an asserted permission unsupported for an unapproved employee, but the same defect need not invalidate permission for an approved employee. References are evaluator-only; independent Python and Z3 checks validate them.

The original corpus has 3,900 families covering access, exceptions/conflict, and bounded obligations. The detector uses 10,675 pairs from 600 fitting families and 2,675 pairs from 150 selection families; separate 75-family roles fit calibration and thresholds. Its warm-started, single-seed weights are fixed before testing. The 600-family test supplies 10,845 pairs and, after deduplication, 11,544 individual inputs, including 4,635 evaluable assertions.

The standalone test crosses six logic categories—access, prohibition, exceptions, conflict, temporal rules, and scope—with six graph shapes, including shared subexpressions, threshold conditions, and conditional selection. The benchmark broadens the chain-based fitting structures and includes equivalent rewrites, graph/query/answer defects, and failed runs. Ten families per combination yield 360 families and 10,080 individual runs; no comparison partner is synthesized. Family isolation does not ensure structural independence: 48 normalized structures recur across original non-test roles. Test fingerprints show no prior match but do not certify semantic independence. Table 6 summarizes construction; Appendix G details selection and input controls.

Held-out claim detection and sensitivity. Original-test scores show a ranking signal: average precision (AP) is 91.88%, above the 65.46% constant-score baseline set by positive prevalence. At raw threshold 0.5, F1 is 81.24%, with 425 false alarms and 668 missed unsupported claims. Family-bootstrap 95% intervals are 80.13–82.31% for F1 and 91.26–92.52% for AP, conditional on this fitted model. Sensitivity uses pairs rather than individual claims: accuracy falls from 99.85% on development to 63.89% on test, with test macro-F1 of 62.98%.

On the standalone test, 2,626 inputs exceed the 640-node IR limit; 1,004 of 3,974 evaluable claims remain unscored. Among scored claims, F1 is 35.52%; AP is 56.56%, below the 66.26% constant-score baseline. Recall is 20.20% when unscored positives count as misses. The post-test capacity diagnostic recovers all missing scores but raises recall on the same positives only to 22.54%. Full coverage therefore does not resolve detection failure.

Development-selected thresholds produce high false-positive rates on both test populations. Repair and replay establish contribution only under the tested interventions; earlier origins remain unresolved. Appendix E gives the full calibration and repair results.

8 Limitations and Conclusion

Automatic detection must distinguish source support from proof against a candidate program. Held-out results from one warm-started model show a ranking signal but limited transfer. The checkpoint extension has development evidence only. Controlled renderings, unrecorded construction modes, and repair-dependent contribution labels limit conclusions about natural policies and attribution. These results establish neither architectural superiority nor human-validated attribution.

The learned scores can prioritize review but do not certify support. Ranking, decision thresholds, coverage, and validated contribution remain separate evaluation targets.

REPRODUCIBILITY STATEMENT

The supplement specifies bounded semantics, proofs, finite constructions, architecture, input views, losses, population definitions, and evaluation settings. It includes aggregate results, confidence intervals, calibration and coverage curves, arithmetic checks, and scripts that reproduce the reported tables and figures. Detector weights and thresholds were fixed before held-out evaluation; the post-test capacity check is reported separately. Source policies, full prediction records, model weights, and the training corpus are not redistributed in this package. The supplied artifacts support checking numerical calculations and rebuilding the manuscript, rather than reproducing training. Model and optimization details appear in Appendix D.

ETHICS STATEMENT

Incorrect policy conclusions can grant unauthorized access or deny legitimate access. This work studies both directions and does not treat abstention as proof of safety. Use of human-authored policies requires appropriate authorization and protection of personal or confidential information. Learned findings should remain separate from established proof outcomes. The method is not validated for operational enforcement, and this manuscript makes no deployment claim.

AI USE STATEMENT

Generative AI assisted with manuscript drafting and editing, literature review and controlled-data development, GPU job coordination, and preparation of architecture diagrams and result plots.

REFERENCES

Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1612.00410.

Philip Arthur, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. Semantic parsing of ambiguous input through paraphrasing and verification. *Transactions of the Association for Computational Linguistics*, 3:571–584, 2015. doi: 10.1162/tacl_a_00159. URL https://aclanthology.org/Q15-1041/.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-Verification Reduces Hallucination in Large Language Models, 2023. URL https://arxiv.org/abs/2309.11495v2. Revision date: 2023/09/25.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024. doi: 10.1038/s41586-024-07421-0. URL https://www.nature.com/articles/s41586-024-07421-0.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/4a8423d5e91fda00bb7e46540e2b0cf1-Abstract.html.

IBM Granite Team. Granite-4.0-H-350M-Base model card. https://huggingface.co/ibm-granite/granite-4.0-h-350m-base, 2025. Accessed 22 September 2026; pinned model revision recorded in Appendix D.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Sarkar Snigdha Sarathi Das, Ranran Haoran Zhang, Wenpeng Yin, and Rui Zhang. Efficient PRM Training Data Synthesis via Formal Verification, 2025. URL https://arxiv.org/abs/2505.15960v3. Revision date: 2026/04/08.

Linggang Kong, Lei Wu, Yunlong Zhang, Xiaofeng Zhong, Zhen Wang, Yongjie Wang, and Yao Pan. CausalGaze: Unveiling Hallucinations via Counterfactual Graph Intervention in Large Language Models, 2026. URL https://arxiv.org/abs/2604.11087v3. Revision date: 2026/05/11.

Jannik Kossen, Jiatong Han, Muhammed Razzak, Lisa Schut, Shreshth Malik, and Yarin Gal. Semantic entropy probes: Robust and cheap hallucination detection in LLMs, 2024. URL https://arxiv.org/abs/2406.15927v1. Revision date: 2024/06/22.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3835–3845, 2019. URL https://proceedings.mlr.press/v97/li19d.html.

Potsawee Manakul, Adian Liusie, and Mark Gales. SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9004–9017, 2023. doi: 10.18653/v1/2023.emnlp-main.557. URL https://aclanthology.org/2023.emnlp-main.557/.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12076–12100, 2023. doi: 10.18653/v1/2023.emnlp-main.741. URL https://aclanthology.org/2023.emnlp-main.741/.

Adam Mondl, Matthew Maisel, and John H. Brock. Autoformalization of Agent Instructions into Policy-as-Code, 2026. URL https://arxiv.org/abs/2606.26649v1. Revision date: 2026/06/25.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10862–10878, 2024. doi: 10.18653/v1/2024.acl-long.585. URL https://aclanthology.org/2024.acl-long.585/.

Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell. Information-theoretic probing for linguistic structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4609–4622, 2020. doi: 10.18653/v1/2020.acl-main.420. URL https://aclanthology.org/2020.acl-main.420/.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.12454.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Soumya Sanyal, Harman Singh, and Xiang Ren. FaiRR: Faithful and Robust Deductive Reasoning over Natural Language, 2022. URL https://arxiv.org/abs/2203.10261v1. Revision date: 2022/03/19.

David Smith Sundarsingh, Jun Wang, Jyotirmoy V. Deshmukh, and Yiannis Kantaros. ConformalNL2LTL: Translating natural language instructions into temporal logic formulas with conformal correctness guarantees, 2025. URL https://arxiv.org/abs/2504.21022v2.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language, 2020. URL https://arxiv.org/abs/2012.13048v2. Revision date: 2021/06/03.

Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method, 2000. URL https://arxiv.org/abs/physics/0004057.

Elena Voita and Ivan Titov. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 183–196, 2020. doi: 10.18653/v1/2020.emnlp-main.14. URL https://aclanthology.org/2020.emnlp-main.14/.

A PROOFS, ASSUMPTIONS, AND LIMITS

This appendix gives the assumptions and proofs behind the information analysis. Each result concerns a fixed population with complete reference outcomes. The analysis distinguishes recovering an outcome, executing it correctly, and asserting a claim supported by the source.

A.1 COLLISION AND SELECTIVE PREDICTION

A representation collision occurs when source cases requiring different outcomes produce the same observed graph and query, not merely the same executed outcome. A predictor restricted to those observations cannot always distinguish the cases. The following result states its minimum classification error and the effect of allowing abstention.

Proposition 1 (Collision and selective prediction)**.** *For finite reference outcomes, a predictor observing only $Z = (G, Q)$ and independent randomness has minimum error $\mathbb{E}e(Z)$, where $e(Z) = 1 - \max_y \Pr(Y^* = y \mid Z)$.*

Here $e(Z)$ is the minimum error conditional on observation Z. A selector uses the same observation to answer with probability $\alpha(Z)$ and otherwise abstains (Geifman & El-Yaniv, 2017). Its randomness is conditionally independent of Y^* given Z. Coverage is the fraction answered, $c = \mathbb{E}\alpha(Z) > 0$, and the minimum error among answered cases is

$$R_{\text{sel}}(\alpha) = \frac{\mathbb{E}[\alpha(Z)e(Z)]}{c}. \tag{8}$$

A simple mixture makes the coverage tradeoff explicit. Suppose an observable subset of probability mass m contains balanced binary collisions, while the remaining observations have deterministic labels. At coverage $c > 0$, the minimum accepted-case error is $\max(0, c - 1 + m)/(2c)$. This calculation uses the additional mixture assumptions; it is not a consequence of every representation collision.

Proof of Proposition 1. Condition on $Z = z = (g, q)$. Returning class y incurs error $1 - p(y \mid z)$, minimized by a posterior mode. Randomized predictions are convex combinations of these errors. Thus the best forced error is $e(z) = 1 - \max_y p(y \mid z)$; integrating gives $\mathbb{E}e(Z)$. Let acceptance be generated with probability $\alpha(z)$ using randomness conditionally independent of Y^* given Z. Conditioning on acceptance therefore does not change $p(y \mid z)$. Accepted error mass is at least $\mathbb{E}[\alpha e]$, attained by the posterior-mode predictor. Dividing by $c = \mathbb{E}\alpha > 0$ proves Equation 8.

In the stated mixture, the distinguishable deterministic subset has mass $1 - m$ and error zero, and the balanced-collision subset has mass m and error $1/2$. If $c \leq 1 - m$, allocate acceptance to the deterministic subset and obtain zero error. If $c > 1 - m$, at least $c - (1 - m)$ acceptance mass lies in the collision subset, contributing half that mass in errors. Accepting all deterministic cases and an appropriate fraction of the collision subset attains $[c - 1 + m]/(2c)$. Randomization permits fractional coverage when observations have atoms. At $c = 0$, selective risk is undefined, not zero; the system makes no predictions. □

For two equally likely policies with the same graph and query but opposite binary outcomes, the residual entropy is one bit. A selector restricted to those observations can make fewer commitments, but cannot reduce the error rate within the accepted part of that collision group. A source-aware monitor has additional inputs and is outside this bound. Classification error can mean missed permission, a mistaken outcome subtype, or unsupported permission. A hallucination finding additionally requires an asserted claim with label one under Equation 2.

A.2 RECOVERABILITY WITHOUT FAITHFUL EXECUTION

Proposition 2 (Recoverability is not fidelity)**.** *An executable encoding can retain every reference outcome ($E^G = 0$) while executing every decision incorrectly.*

Proof of Proposition 2. Consider a binary special case with fixed context: $Q \in \{1, \ldots, m\}$ indexes a query and $Y^* \in \{0, 1\}$ is its answer. Let $B \in \{0, 1\}^m$ be the complete policy answer vector.

Define $Y^* = B_Q$, $G = \mathbf{1} - B$, and literal interpreter $V(G, Q) = G_Q$. This bijection permits exact recovery of B, so $H(Y^* \mid G, Q) = 0$, while $V(G, Q) \neq Y^*$ always. Each lookup vector has a finite DAG representation. The related incidental-structure construction in Appendix B shows that zero residual entropy also permits deleted constraints, without requiring explicit source identifiers. □

A semantic graph has operators with fixed executable meanings. A decoder may recover the reference outcome even when those operators compute the wrong answer. Grouping behaviorally equivalent programs does not ensure agreement with the reference either. Candidate-execution prediction, reference-outcome probing, and claim-support assessment therefore measure different properties.

A.3 Conditional information with multiple inputs

A checkpoint may combine several predecessors or reread the source. To interpret its information change, we must account for both kinds of input. Let U_ℓ collect the predecessor artifacts and $W_\ell = G_\ell$ denote the output. Conditional mutual information I measures the information one variable supplies about Y^* after the conditioning variables are known. The change in residual uncertainty satisfies

$$H(Y^* \mid W_\ell, Q) - H(Y^* \mid U_\ell, Q) = I(Y^*; U_\ell \mid W_\ell, Q) - I(Y^*; W_\ell \mid U_\ell, Q). \tag{9}$$

The first term measures information available in the predecessors but not the output; the second measures information available in the output beyond those predecessors. Both condition on the same query and context Q. With $U = U_\ell$ and $W = W_\ell$, the chain rule gives

$$I(Y^*; U \mid W, Q) = H(Y^* \mid W, Q) - H(Y^* \mid U, W, Q),$$
$$I(Y^*; W \mid U, Q) = H(Y^* \mid U, Q) - H(Y^* \mid U, W, Q).$$

Subtracting proves Equation 9. If $W = f(U, Q)$, the second term is zero. If the stage also consumes fresh input S, such as reread source text, $W = f(U, Q, S)$ and that term need not vanish. Comparing with the complete input tuple (U, S) is valid, but defines a different conditional entropy.

Every comparison uses the same reference target and population. If new evidence changes the context, both sides must be evaluated under that context or treated as different tasks. Failed and missing outputs remain explicit symbols. Restricting each layer to its own successful cases would change the population and invalidate the comparison.

A.4 Information in a graph versus its embedding

The executable graph and its learned embedding are different observations. To compare them, use one joint distribution of policy families, verifier randomness, and queries with complete reference targets. Graph-only features exclude source passages, source-conditioned states, provenance, family identifiers, diagnostics, and labels. This prevents direct access to those fields, but does not remove correlations carried by incidental graph structure.

For a checkpoint record G_ℓ, define its residual outcome uncertainty as

$$E_\ell = H(Y^* \mid G_\ell, Q). \tag{10}$$

It averages uncertainty about the complete reference outcome over the same population.

Fix ϕ before evaluation, use deterministic graph-only features, and let $Z_\phi = z_\phi(G, Q)$. Because Z_ϕ is a function of (G, Q), $H(Y^* \mid G, Z_\phi, Q) = H(Y^* \mid G, Q)$. Hence

$$I(Y^*; G \mid Z_\phi, Q) = H(Y^* \mid Z_\phi, Q) - H(Y^* \mid G, Q) = E^z - E^G \geq 0. \tag{11}$$

This conditional data-processing identity measures information lost by the evaluated map. The encoder may be learned before it is fixed; training can make retained information easier for a restricted predictor to use. The identity requires the evaluated embedding to be a deterministic function of (G, Q). Additional source or label inputs would define a different observation.

With ϕ fixed and readout parameters ψ fitted separately, population probe loss is $E^z + \mathbb{E}_{Z_\phi, Q}\mathsf{KL}_2(p(\cdot \mid Z_\phi, Q) \| \rho_\psi(\cdot \mid Z_\phi, Q))$. The second term is the readout's approximation and fitting gap. Graph and embedding probes can have different gaps on the same population, so their loss difference does not directly measure information loss. Successful prediction from either view also leaves execution correctness to be checked separately.

A.5 PROBE CROSS-ENTROPY AND DESCRIPTION LENGTH

A probe estimates how well a specified predictor can recover reference outcomes. Let $p(y \mid g, q)$ be the population conditional distribution and $\rho_\psi(y \mid g, q)$ the probe's normalized prediction, with parameters ψ fitted on separate families. Assume the probe assigns positive probability wherever p is positive. Its population cross-entropy decomposes as

$$\mathbb{E}[-\log_2 \rho_\psi(Y^* \mid G, Q)] = H(Y^* \mid G, Q) + \mathbb{E}_{G,Q}\mathsf{KL}_2(p(\cdot \mid G, Q) \| \rho_\psi(\cdot \mid G, Q)). \tag{12}$$

The first term is uncertainty remaining after observing the graph and query. The second is the nonnegative Kullback–Leibler divergence between the true and predicted conditional distributions, reflecting approximation and fitting error. Base-two logarithms express both terms in bits. A held-out mean estimates their sum over the fixed population.

For any positive normalized ρ_ψ on the target support,

$$-\sum_y p(y \mid g, q) \log_2 \rho_\psi(y \mid g, q) = -\sum_y p(y \mid g, q) \log_2 p(y \mid g, q) + \sum_y p(y \mid g, q) \log_2 \frac{p(y \mid g, q)}{\rho_\psi(y \mid g, q)}.$$

Average over (G, Q) to obtain Equation 12. This is a population upper bound; finite held-out estimates can lie below population entropy by sampling variation. Differences between probes combine entropy, approximation, and fitting effects. To ensure positive support, a probe can use $\rho'_\psi = (1 - \epsilon)\rho_\psi + \epsilon/K$, where K is the number of target classes and $0 < \epsilon < 1$ is a declared smoothing constant. This specifies an admissible predictor, not a measured result.

Learning curves, capacity-matched readouts, and shuffled-label controls help assess fitting effects. Independent probing also requires a fixed representation and disjoint fitting and evaluation families. These conditions specify how to use the diagnostic; independent outcome-probing results are not reported in this study.

Online minimum description length (MDL) measures the bits needed to encode labels under a sequential learning protocol. Group whole families into predeclared blocks, encode the first block with a uniform K-class code, and encode each later block with a probe fitted only on earlier blocks:

$$L_{\text{online}} = n_1 \log_2 K + \sum_{k=2}^{K_b} \sum_{i \in B_k} -\log_2 \rho_{\psi_{<k}}(y_i \mid g_i, q_i). \tag{13}$$

Here B_k contains the example indices in block k, K_b is the number of blocks, and n_1 counts first-block labels. The targets y_i are reference outcomes. Graphs and queries are shared side information; the learning procedure and block order are fixed. No paraphrase or mutation of an encoded family may appear in an earlier fitting block. This code length measures predictive effort under the chosen protocol, separately from graph size, exact conditional entropy, and logical correctness. No MDL measurements are reported.

A.6 WHAT A DISCREPANCY WITNESS PROVES

Proposition 3 (Bounded discrepancy witness)**.** *Fix validated policy and query interpretations, context, opposition mapping, finite domains, and completed sound evaluations. If an admissible query/context case Q yields $V(G, Q) \neq V(P^*, Q)$, the candidate and reference disagree on that bounded instance. A permission witness with $s_G = 1, s^* = 0$ establishes candidate support for permission that the reference does not support at that case. It does not, by itself, establish strict expansion of the permission region. Claim emission and defect origin require separate evidence.*

Proof of Proposition 3. Bounded semantic equivalence requires equality for every admissible query and context in the declared domain. Two completed sound evaluations with different outcomes provide a counterexample. For permission, $(s_G, s^*) = (1, 0)$ identifies a case supported by the candidate but not the reference. This comparison establishes neither what an answer asserted nor which transformation introduced the discrepancy. □

A witness is meaningful only together with its source/reference versions, interpreted query, context, finite domains, and evaluation statuses. With ambiguous policies or incomplete observations, an unconditional finding requires the same claimed discrepancy across all admissible reference scenarios. Otherwise the finding is conditional on the chosen interpretation or completion. Checking a sampled query set establishes agreement only on that set; a failed or unfinished check leaves the comparison unresolved. A discrepancy witness establishes disagreement, while repair and replay address contribution to a claim.

A.7 Reference alignment, query comparisons, and notation

Figure 2 compares reference evaluation, candidate execution, and prediction from candidate embeddings under the same context C. Figure 1 defines the reference outcome. Solid arrows carry processing inputs; dashed links compare results without supplying inputs or gradients. Route agreement, or commutativity, requires the routes to answer the same question and return the same outcome throughout the declared domain.

From scenario outcomes to a common target. The validated query q^* fixes proposition $a(q^*)$, including its bound arguments, modality, polarity, scope, and time. Let $\mathcal{S}(X, C) = \mathcal{P}^*(X) \times \Omega(C)$ be the nonempty set of reference scenarios and $\mathcal{Y} = \{0, 1\}^2$ the outcome space in Figure 1. When every scenario check is completed, the query target is

$$Y^*(X, Q) = y \in \mathcal{Y} \quad \Longleftrightarrow \quad Y_{P,c}(a(q^*)) = y \quad \forall (P, c) \in \mathcal{S}(X, C).$$

If outcomes disagree or a required check is unavailable, Y^* remains unresolved. By contrast, $(0, 0)$ records completed checks establishing neither support nor refutation, and $(1, 1)$ records a definite conflict. The pair (s, r) records support and refutation for the queried proposition. Pending, fulfilled, and violated describe an obligation's temporal status at the evaluation time; a status query applies (s, r) to that status proposition.

The shorthand Y^* suppresses (X, Q); $Y^*(Q)$ suppresses fixed X. With one validated interpretation P^* and agreement over context completions, $V(P^*, Q)$ denotes the same reference result. This notation does not require reference and candidate evaluators to share an implementation.

Agreement on the complete outcome is stricter than agreement on a claim label. For a permission proposition, reference states $(0, 0)$ and $(0, 1)$ both classify an answer asserting permission as unsupported; the latter also establishes prohibition. Neither state is itself a hallucination: the label applies to what the answer asserts. An answer asserting prohibition must instead be checked relative to that proposition. If admissible scenarios yield these two states, the permission-claim label agrees but refutation does not, leaving no unique complete-outcome target. Conversely, unanimous $(1, 1)$ establishes a definite conflict. Presenting permission as settled ($u = 1$) misrepresents that conflict; explicitly acknowledging it ($u = 0$) does not.

Construction, embeddings, and readout. With other run inputs fixed, the joint maps are

$$(G, \widehat{q}) = \mathrm{PV}(X, q^{\mathrm{txt}}, C), \qquad Z_\phi^{\mathrm{cand}} = E_\phi(G, \widehat{q}, C) = (z_\phi^P, z_\phi^Q),$$
$$Y_{\mathrm{exec}} = V(G, (\widehat{q}, C)), \qquad \widehat{Y}_\psi = \arg\max_{y \in \mathcal{Y}} \rho_\psi(y \mid Z_\phi^{\mathrm{cand}}, C).$$

Ties in the readout use a fixed rule. Parameters ϕ belong to the encoders and ψ to the probability readout. The two construction arrows identify outputs of one process: query formalization can depend on the policy. Likewise, the two embedding arrows denote components of one candidate-record map. The superscripts P, Q name program and query branches; they do not assert independent maps or shared program/query weights. Context supplies evidence, time, and completeness to all three routes.

This analytical readout targets the original question's outcome, even when its query features come from a corrupted candidate. Lost query information can therefore make the target unrecoverable. For this candidate-input route, validated reference meanings, alignment findings, and labels remain evaluation-side information. The readout is distinct from the paired-sensitivity head and from an emitted answer; the diagram specifies a comparison, not an additional measured probing result.

Compare targets before comparing outcome values. Write $\tau^* = \tau(q^*)$ and $\widehat{\tau} = \tau(\widehat{q})$ for normalized semantic target descriptors. A descriptor records the proposition's predicate or action, normative modality, polarity, bound actor/resource arguments, scope, and temporal interval. These fields must be established from the interpreted query, not copied from a provenance ID or inferred from agreement of output bits. Renamed variables and equivalent serializations can denote the same target. If a descriptor or its correspondence cannot be established, target comparison is unresolved.

For established descriptors, let $\delta_{\text{target}} = \mathbf{1}[\widehat{\tau} \neq \tau^*]$. Value disagreement is defined only for aligned, completed comparisons:

$$\delta_{\text{exec}} = \begin{cases} \mathbf{1}[Y_{\text{exec}} \neq Y^*], & \delta_{\text{target}} = 0 \text{ and both outcomes are complete,} \\ \mathsf{unresolved}, & \text{otherwise.} \end{cases}$$

A known target mismatch remains a reported query defect, even when no aligned value comparison exists. A failed candidate execution is reported as a failure, not a semantic pair; it counts as a mismatch in the separately defined failure-inclusive D_μ protocol. Missing or ambiguous reference outcomes cannot be assigned a negative defect label. Report attempted cases, target mismatches, unresolved comparisons, completed value comparisons, and execution failures separately, so conditioning on alignment cannot hide query errors.

Consider "must the employee submit by day 5?" and "may the employee submit by day 5?" Both can return $(1, 0)$, yet the obligation and permission targets differ: $\delta_{\text{target}} = 1$. Matching bits do not repair that query defect. Matching descriptors, in turn, identify the same proposition but do not guarantee a correct query computation. A compiler can retain the permission target while replacing its test by a constant. Detecting that error requires comparisons across the declared cases, not one coincidentally correct answer. Policy and query errors can also cancel numerically without establishing faithful interpretation.

The prediction target remains $Y^*(X, Q)$. When it and the readout output are available, $\delta_{\text{probe}} = \mathbf{1}[\widehat{Y}_\psi \neq Y^*]$ is defined even if the candidate query target is wrong: the readout is defined to predict the original target. A correct prediction may coexist with a query defect or incorrect execution. Complete route agreement requires valid reference checks, aligned candidate targets, completed execution, and agreement of the relevant outcomes on every case in the declared domain. No sampled agreement establishes an unrestricted guarantee.

Separate policy and query controls. For a single trusted interpretation P^*, $\delta_P = \mathbf{1}[V(G, (q^*, C)) \neq Y^*]$ tests the candidate program while holding the validated question fixed, whenever outcomes are complete. The query control executes $V(P^*, (\widehat{q}, C))$ and first checks δ_{target}. Only aligned, completed cases admit the value comparison $\delta_Q = \mathbf{1}[V(P^*, (\widehat{q}, C)) \neq Y^*]$. Mismatch, ambiguity, and failure are retained separately. With multiple admissible references, apply these controls across scenarios and require agreement for a definitive conclusion. All such controls are reference-assisted evaluation, not deployment inputs. Repair and replay remain necessary for intervention-supported contribution to a completed claim.

Two access regimes. The reference-assisted regime supplies a trusted formal interpretation and permits exact bounded comparison. Its evaluation includes check completion, cost, and the usefulness of any localized discrepancy. The deployment-like regime supplies source text, candidate artifacts, and recorded runs to the detector; trusted interpretations remain evaluator-only. A learned score in the second regime does not acquire the guarantees of a completed exact check in the first.

Notation and information boundaries. The joint candidate embedding Z_ϕ^{cand} differs from the fixed-query diagnostic $Z_\phi = z_\phi(G, Q)$ in Equation 5: the latter supplies validated q^* specifically to isolate policy-graph information. Its information bound cannot silently be transferred to a population or input view using $\widehat{q}$. Within each diagnostic, hold the outcome target, input variables, and population fixed. Failed or missing candidate records require explicit symbols; they are not silently discarded. Reference artifacts remain outside operational detector inference.

We use $r_{P,c}(a)$ only for logical refutation and ρ_ψ for a learned probability readout. The encoder map E_ϕ and residual entropies E^G, E^z are distinct objects; the latter are measured in bits. A question's text q^{txt}, validated meaning q^*, candidate meaning $\widehat{q}$, and query/context pair Q remain separate.

Query targets $a(q^*)$ must also be distinguished from the actual emitted claim a and assertion mode u used by $\mathsf{Hall}(a, u)$.

During detector learning, gradients pass through neural encoders and heads while PV, discrete artifacts, symbolic V, target validation, and labels remain fixed. The optional objective in Appendix D uses a frozen readout that can still transmit gradients to its input embeddings. This differs from a detached diagnostic probe. The complement construction can give $\delta_{\text{probe}} = 0$ and $\delta_{\text{exec}} = 1$ for the same aligned target. Conversely, a correctly executing program paired with an incorrect readout can give $\delta_{\text{exec}} = 0$ and $\delta_{\text{probe}} = 1$. Changing the readout alone does not change execution correctness. Claim support and contribution require their own checks.

Terminology used throughout the paper. An executable *program* is the discrete semantic artifact G; a *graph embedding* is a learned continuous representation such as $z_\phi(G, Q)$. Text-backbone features and source-aware states are separate representations. *Semantic sensitivity* describes behavior under decision-relevant edits; *differentiability* describes derivatives through neural computations. *Execution disagreement*, also called executable distortion, compares candidate and reference outcomes. *Recoverability* concerns predicting the reference outcome from a specified input view.

A *defect* is a failed semantic preservation requirement. *Hallucination* concerns a completed claim with missing source support or conflict presented as settled. State $(0, 0)$ means neither support nor refutation, not a failed check or an explicit prohibition. A *checkpoint* records one processing event; a *run* is the verifier execution and its artifacts, while its *trace* is the recorded event sequence and dependencies. *Localization* identifies a suspected defect location; *intervention-supported attribution* requires validated repair and replay. The paired "claim effect" target is a classification label and carries no causal meaning.

A *claim score* is an uncalibrated neural output. Empirical *risk* is a stated error or hallucination frequency on a declared accepted population. The *candidate-execution head* is trained with the detector; a *diagnostic probe* assesses reference-outcome recoverability with the representation fixed. The optional *frozen readout* retains input gradients and therefore serves a different training role.

B Exact constructions and worked policy examples

The examples below separate retained information, executed behavior, and asserted claims. All values follow from exact enumeration of finite cases; they are analytical controls rather than measured model performance.

B.1 Two-bit population

Consider a complete binary special case with fixed context. Let $B \in \{0, 1\}^2$ be uniform and let an independent uniform query index $Q \in \{0, 1\}$ select the reference answer $Y^* = B_Q$. Each of the eight policy/query cases has probability $1/8$. The representation and its literal executor are defined together:

Encoding	Stored vector	Literal executor
Faithful	(B_0, B_1)	Return coordinate Q
One bit lost	$(B_0, 1)$	Return coordinate Q
Constant allow	$(1, 1)$	Return one
Complemented	$(1 - B_0, 1 - B_1)$	Return coordinate Q

The faithful encoding has zero residual entropy, minimum prediction error (Bayes error), and execution disagreement. Replacing B_1 by one removes the queried bit when $Q = 1$, leaving average residual entropy $1/2$. Execution is wrong exactly when $Q = 1, B_1 = 0$, so its disagreement rate and Bayes error are both $1/4$. Constant allow leaves one bit of uncertainty and gives error $1/2$ under both measures. Complementing is invertible: residual entropy and Bayes error are zero, although every literal execution is wrong.

In this special case, one means permission and zero means explicit prohibition, corresponding to support/refutation states $(1, 0)$ and $(0, 1)$. These are outcome values, not hallucination labels. Define

unsupported-permission mass as $D^+ = \Pr(V(G,Q) = 1, Y^* = 0)$. Its values are $0, 1/4, 1/2, 1/2$, respectively. It counts added permission cases without requiring strict region expansion. The complemented encoding also removes permission on half the cases, so its permission region does not strictly expand.

Assume the answer renderer asserts permission when execution returns one and otherwise abstains. The mass of emitted unsupported claims then equals D^+. Permission coverage is $1/2, 3/4, 1, 1/2$, while the unsupported fraction among emitted claims is $0, 1/3, 1/2, 1$. The unconditional mass and risk among emitted claims have different denominators. These renderer-specific rates also differ from the optimal outcome-prediction error in Appendix A.1.

Exact enumeration verifies these values and the incidental-structure and paired examples below. Additional checks cover the selective-risk formula, the finite-candidate gradient by central differences, and the graph-to-embedding information calculation. These analytical checks do not evaluate a trained detector.

B.2 Deleted constraint with source-identifying structure

Incidental structure can preserve a source distinction after a governing constraint has been lost. Let $B \sim \text{Bernoulli}(1/2)$ select between two policies at one fixed query: $B = 0$ requires approval, whereas $B = 1$ authorizes unconditionally. Approval is known absent and fact t is known true. Here $Y^* = B$ records permission support, with full outcomes $(0,0)$ and $(1,0)$, respectively; zero does not imply prohibition. A defective encoder always authorizes, using t when $B = 0$ and $t \wedge t$ when $B = 1$. Both graphs execute as true, but their structures identify which policy was supplied. Hence $H(Y^* \mid G, Q) = 0$, while execution disagreement and unsupported-permission mass are each $1/2$.

No source passage or family identifier is copied. Normalizing away the redundant conjunction makes the graphs collide and raises residual entropy to one bit, while execution disagreement remains $1/2$. Canonical serialization need not perform this algebraic normalization. An entropy change therefore cannot, by itself, be interpreted as a change in correctness.

B.3 Answer regions, insensitivity, and claim support

An answer region contains all cases assigned a particular outcome. Fix reference interpretation P^* and a bounded domain $\mathcal{D}$ of query/context cases with complete reference outcomes. Equation 3 compares candidate and reference regions on this common domain. The set $\mathcal{R}_G(y) \setminus \mathcal{R}^*(y)$ contains cases newly assigned outcome y; $\mathcal{R}^*(y) \setminus \mathcal{R}_G(y)$ contains cases that lose it. Strict expansion requires at least one added case and no removed cases. Other errors can add and remove cases simultaneously, so region size alone does not establish expansion. Failed executions retain their status within the declared domain but belong to no semantic outcome region.

For an exact construction, fix the question of employee access. Contexts vary Boolean Employee e and Approval b over all four combinations. The sole reference rule is $e \wedge b \Rightarrow \text{Permit}$, with no prohibition. A defective candidate uses $e \Rightarrow \text{Permit}$. Using local outcome aliases $U = (0,0)$ for neither support nor refutation and $P = (1,0)$ for support only,

Employee	Approval	Reference outcome	Candidate outcome
0	0	U	U
0	1	U	U
1	0	U	P
1	1	P	P

The reference permits only employees with approval; the candidate also permits employees without approval. For an employee ($e = 1$), approval changes the reference decision but no longer changes the candidate decision. The omitted condition has therefore lost its semantic effect. A renderer that asserts permission on candidate outcome P emits an unsupported claim in context $(e, b) = (1, 0)$. If it abstains there, the defect remains but no hallucinated claim is emitted. Reference outcome U means permission is unsupported, not explicitly prohibited.

The paired evaluation uses a complementary view: hold Q fixed and change policy text. For a bounded collection $\mathcal{X}$ of policies with complete reference outcomes, define $F_Q(X) = V(G_{\mathrm{PV}}(X), Q)$ and $F_Q^*(X) = Y^*(X, Q)$. Their inverse images $F_Q^{-1}(y)$ and $(F_Q^*)^{-1}(y)$ are regions in policy space, whereas Equation 3 uses query/context space for a fixed policy. If a restrictive policy and a policy without its approval requirement both generate unconditional employee permission, they share outcome $(1, 0)$ at the unapproved-employee query even though their reference outcomes differ. Equation 4 detects this lost behavioral distinction. Equal executed outcomes need not come from identical graphs, so this alone does not establish the representation collision required by Proposition 1. One pair also does not establish global expansion, region size, or insensitivity throughout the domain. The reported experiment measures five-class policy-pair classification, not answer-region volumes.

The detector learns from such discrete contrasts. Its shared encoders, symmetric and directional features, and prediction heads are differentiable with respect to trainable parameters. A supervised loss therefore supplies gradients for recognizing source/program discrepancies. The compiler, candidate graph, executed outcomes, and labels stay fixed during each update. A small parameter gradient, a small embedding distance, or a graph collision alone does not establish hallucination. Claim support and assertion status remain the defining tests; source-aware inputs can supply distinctions absent from identical graph-only inputs.

B.4 Semantic-sensitivity controls

Use the reference ordered pair (y, y') and candidate pair (v, v') for a fixed query/context. All outcomes are complete. The local aliases $U = (0, 0)$ and $P = (1, 0)$ mean neither support nor refutation and support-only permission, respectively. These aliases are outcome values, not policy programs. Compute $S_{\mathrm{lost}} = \mathbf{1}[y \neq y']\mathbf{1}[v = v']$ and ordered-pair error $\mathbf{1}[(v, v') \neq (y, y')]$.

Case	Reference pair	Candidate pair	S_{lost}	Pair error
Equivalent paraphrase	(P, P)	(P, P)	0	0
Required change preserved	(U, P)	(U, P)	0	0
Required change lost	(U, P)	(P, P)	1	1
Wrong-direction change	(U, P)	(P, U)	0	1
Edit irrelevant to query	(P, P)	(P, P)	0	0
Spurious behavioral change	(P, P)	(U, P)	0	1

The lost-change row is realized by a restrictive policy requiring approval and a permissive variant without that requirement, for an unapproved employee. If both are encoded as unconditional permission, the restrictive instance has an unsupported-permission witness. With a renderer emitting permission, that instance yields an unsupported claim; with a renderer abstaining, the same defect yields no emitted hallucination. A query with approval already present is insensitive to this edit and must not receive a lost-distinction label. These constructions test the diagnostic's meaning, not a learned model.

The sensitivity protocol associates source edits with validated references while holding question and context fixed. Every paraphrase and mutation remains within its family's split. Graph-only mutations assess downstream behavior; source-edit preservation requires evaluating the construction process itself. Lost-distinction rates use reference-changing pairs, spurious-change rates use reference-invariant pairs, and ordered-pair error uses all completed pairs. Completion is reported over all attempts.

Comparisons of generated runs additionally require control of generation variability, for example through deterministic decoding or paired random streams. The controlled examples here specify the candidate artifacts directly. Embedding distance alone cannot define semantic collapse: reparameterization can change distances while preserving behavior.

B.5 Policy support, conflict, and temporal status

Approval. Suppose the only authorization rule is $\mathrm{Employee} \wedge \mathrm{Approval} \Rightarrow \mathrm{Permit}$. For an employee without approval, permission has outcome $(0, 0)$. Dropping Approval changes the candidate outcome to $(1, 0)$ and supplies an unsupported-permission witness. The reference does not prohibit

access; it lacks sufficient support to permit it. For an employee who already has approval, the same graph defect leaves the outcome unchanged and does not make the supported permission claim a hallucination.

Exception. When both apply at equal priority, a general permission and a prohibition on restricted records produce outcome $(1, 1)$ for permission. Suppose instead that the source gives the prohibition higher priority and a scoped defeat edge; the reference then returns $(0, 1)$. A candidate omitting that edge returns $(1, 1)$, but an answer asserting permission remains unsupported by the unchanged reference. Scope remains essential: an exception for reading does not override a writing obligation.

Deadline. Fix a source obligation triggered at time 2 and due by time 5, inclusive. An authenticated matching action at time 5 fulfills it. With complete observations and no matching action, it is pending at time 4 and violated at time 6. A gap that could contain an on-time action leaves fulfillment versus violation uncertain. Under complete observations with no action, a candidate that incorrectly extends the deadline to 7 records a pending obligation at time 6; the original day-5 obligation remains violated. This deadline mismatch must be retained when comparing targets. An answer claiming that the original obligation is not yet overdue is unsupported; abstention emits no hallucinated claim.

C Operational bounded semantics

The reference language defines how policy support is computed over finite domains. It keeps normative outcomes, incomplete observations, and execution failures separate. The following rules specify the bounded fragment used in the analytical constructions.

C.1 Grammar, grounding, and observations

Evaluation begins by declaring finite domains for actors, actions, resources, facts, and time. Rule variables are finitely grounded before evaluation; unrestricted quantification, recursive rule inference, epistemic modalities, and unbounded temporal reasoning are excluded. A compact grammar is

$$
\begin{aligned}
b &::= \top \mid \bot \mid f(\bar{e}, t) \mid \neg b \mid b \wedge b \mid b \vee b, \\
n &::= \mathrm{Permit}(\bar{e}) \mid \mathrm{Prohibit}(\bar{e}) \mid \mathrm{Oblige}(\bar{e}, \delta) \mid \mathrm{Exempt}(\bar{e}, \delta), \\
r &::= (\mathrm{id}, b, n, I), \qquad P = (R, \succ, \kappa).
\end{aligned}
$$

Here b is a Boolean condition, n a normative conclusion, and r a rule. The fact $f(\bar{e}, t)$ concerns typed entities $\bar{e}$ at time t. An obligation's nonnegative offset δ determines its deadline. Each rule has an applicability interval I; policy P contains the finite rule set R, acyclic priority relation $\succ$, and scoped defeat relation κ. Rule conditions depend on facts rather than normative conclusions. Normative arguments and opposition mappings are type checked.

Context C records known facts, authenticated actions, evaluation time, and which observation intervals are complete. Missing facts remain unassigned. The finite, nonempty set $\Omega(C)$ contains all complete bounded histories consistent with these records and domain constraints, up to the evaluation time. Completions add no future events. A context admitting no completion is invalid.

Boolean expressions use ordinary truth tables within each completion. Three-valued evaluation is permitted only as a sound shortcut. For example, $f \vee \neg f$ is true in every completion even when a coarse three-valued evaluator returns unknown. Exact completion semantics therefore remains the reference for such shortcuts.

C.2 Priority, support, and opposition

Evaluation first determines which rules apply and which are defeated, then collects support. Priority alone cannot suppress a rule; an admitted scoped defeat edge is also required. At a complete context and grounding, $\mathrm{app}(r)$ is true exactly when the evaluation time lies in I_r and condition b_r is true. Admit $s \ \kappa \ r$ only when $s \succ r$ and the scopes match the relevant grounding. Evaluate the resulting

acyclic relation with higher-priority rules first:

$$\text{active}(r) = \text{app}(r) \wedge \neg \bigvee_{s:\, s \kappa r} \text{active}(s). \tag{14}$$

Only an active scoped defeater suppresses a rule. Defeat does not propagate through an inactive rule: each lower-priority rule must have its own active defeater. Cyclic priority or defeat input is invalid. Equal-priority opposing rules without a defeat edge remain active together. These rules stipulate one bounded prioritized semantics, rather than a general account of deontic logic.

For a ground normative proposition a, support is one exactly when an active rule concludes a; refutation is one exactly when an active rule concludes its declared opposite $\bar{a}$. Obligation conclusions additionally bind the trigger instance. Permission and prohibition oppose each other for the same action and scope; obligation and exemption oppose each other for the same trigger and deadline. Permission does not refute obligation. Factual negation inside a condition is separate from normative opposition. Different deadlines identify different obligation instances unless the reference explicitly relates them.

With missing observations, evaluation retains the set of outcomes across admissible completions, using enumeration or an equivalent complete symbolic check. Uncertainty cannot be replaced by $(0, 0)$. A definitive binary hallucination label requires only agreement on the claim classification; a complete-outcome probe additionally requires agreement on the full semantic target. The probe cohort is fixed before comparing representations.

C.3 Triggered obligations and finite observations

Issuing an obligation and assessing its satisfaction are separate operations. At each designated trigger time t_0, rules are evaluated at that time. An active obligation creates a token $(\bar{e}, t_0, d)$ with deadline $d = t_0 + \delta$. The policy schema specifies trigger matching; a fact remaining true does not create repeated obligations unless repeated triggers are declared. An active exemption with an admitted scoped defeat edge prevents issuance for that instance. The fragment excludes later cancellation and retrospective policy changes.

For an issued token, satisfaction is evaluated from the complete history through time t. A matching authenticated action in $[t_0, \min(t, d)]$ makes the token fulfilled. Without such an action it is pending for $t \leq d$ and violated for $t > d$. Issuance and satisfaction are separate questions; an obligation does not predict that its required action will occur.

The matching relation determines whether one action can satisfy several tokens; default matching uses an obligation-instance identifier. Status propositions are derived from the temporal checks above, rather than from normative rule conclusions. For an issued token, support for a status proposition a is one when the completed history check establishes a; refutation is one when it establishes the declared opposite $\bar{a}$. For example, "unfulfilled at time t" means no on-time matching action has occurred by t; it includes both pending and violated states. Incomplete observations retain the possible statuses across completions. Pending is a temporal state, not an unresolved check.

Selecting no obligations cannot establish that every source-applicable obligation was fulfilled. That conclusion also requires a complete check of which obligations apply. The completeness check is recorded separately from a statement that holds vacuously over the selected set.

C.4 Execution disagreement and error categories

Fix candidate program G, validated reference P^*, and a finite population with complete reference outcomes. Distribution μ assigns nonnegative weights summing to one to its query/context cases. For each Q, candidate outcome $V(G, Q)$ is compared with reference outcome $Y^*(Q)$. A difference in either support/refutation coordinate contributes one; agreement on both contributes zero. The weighted mean $D_\mu(G, P^*)$ is therefore an execution-disagreement rate. Zero means agreement on every case with positive weight, and one means disagreement on every such case. Its scope is this population; emitted-claim support is measured separately.

For a constructed example with ten equally weighted cases, suppose seven candidate executions match, two return different outcomes, and one fails. The failed execution has a distinct failure

outcome, so $D_\mu(G, P^*) = 3/10$. Report the failure rate $1/10$ as well as disagreement among completed executions, $2/9$. Cases without a complete reference outcome cannot receive a mismatch label and are excluded from this reference-complete cohort, with their count reported separately. These fractions illustrate the definition; they are not experimental results.

Evaluation uses the validated query in $Q = (q^*, C)$ to isolate program errors. Using candidate query $\widehat{q}$ instead requires the target-alignment checks in Appendix A.7. A changed question is a query defect even if its output bits match the reference; value disagreement is assessed only for aligned targets.

Execution-error categories. For completed comparisons, unsupported-permission mass is $D^+ = \Pr(s_G(\text{Permit}) = 1, s^*(\text{Permit}) = 0)$, where s_G and s^* denote candidate and reference support. This counts added support without requiring strict expansion of a permission region. Unsupported denial means that the candidate supports Prohibit without reference support for it. Missed permission means that the candidate fails to support Permit when the reference does. A missed permission therefore need not be an asserted prohibition. Conflict suppression and temporal misclassification form additional, potentially overlapping categories. Each rate uses an explicitly stated population.

C.5 Stage semantics and graph contracts

Two graph types carry different meanings. A policy-expression graph represents typed Boolean conditions and grounded normative roots, connected by argument, condition, trigger, scope, and deadline edges. Priority and defeat relations are checked separately. An event graph connects recorded processing steps through their dependencies, with each recorded stage invocation represented as a distinct event. Canonical identifiers link records but are not learned identity features. Source spans, versions, and correspondence confidence remain outside graph-only features.

Each stage has a preservation obligation. Source interpretation retains sufficient versus necessary conditions; graph construction retains the interpreted assertions; compilation retains bounded execution behavior. Query formalization preserves proposition, entities, time, and modality. Selection retains applicable support and opposition, while proof-to-answer conversion retains the checked outcome and completion status. Renaming nodes, reordering commutative inputs, and equivalent rewrites must preserve execution.

Cross-stage analysis uses typed snapshots: assertions, programs, selected rules, proofs, or extracted answer propositions. An adapter may change their format while preserving the reference target. Testing an intermediate artifact by execution additionally requires a declared trusted continuation; otherwise only its local preservation requirement can be checked. Local contract violations and source-relative errors remain separate because a stage may inherit, introduce, or correct a defect.

D Architecture, objectives, and evidence requirements

This section specifies the frozen detector used for held-out evaluation, its training procedure, and the evidence needed for detection and replay. Appendix D.8 describes the checkpoint extension used in the supplementary development studies. Two analytical objectives illustrate alternative gradient paths; neither is part of the reported detector training.

D.1 Feature views and graph encoders

Policy Verifier constructs and executes the candidate programs; detector training does not update it. The frozen text backbone is `ibm-granite/granite-4.0-h-350m-base`, with an immutable model revision recorded in the supplementary configuration. The backbone supplies 768-dimensional causal token features, so a token feature does not automatically encode later source text. Source-aware and semantic-record views are encoded independently, with recurrent state reset between inputs. Text is processed in chunks of at most 1,024 tokens within an 8,192-token budget. Inputs exceeding the budget remain unscored; content is not silently truncated. Chunking alone does not provide each token with context from all chunks.

Source units are token groups separated by punctuation or newlines. Groups longer than 64 tokens are split. An input requiring more than 128 groups remains unscored. Accepted inputs retain every non-special source token, although the groups are not independently validated semantic clauses.

Separate encoders represent executable programs and intermediate-representation (IR) records. Each has four blocks of width 256, eight attention heads, and dropout 0.1. Corresponding encoders share weights across the two runs. Each block combines typed directed attention, local relation messages, feed-forward updates, and query conditioning. Structural biases represent forward/reverse distance capped at eight, depth, and in/out degree capped at 16. Query-conditioned attention pools node features into a 256-dimensional embedding.

IR records have a 640-node budget and serialization depth eight. The graph-only view admits executable records, query, and context. It excludes source passages, reference programs, provenance identifiers, labels, and construction metadata. These input restrictions define which information the embedding can access, independently of which losses train its parameters.

D.2 Source alignment and individual-claim inference

Source alignment looks for requirements without counterparts in the executable graph. Two width-256, eight-head blocks combine source self-attention, source-to-node attention, node-to-source attention, and residual feed-forward updates. Question and context condition copies of the source/node features. A learned null node lets a source unit remain unmatched instead of forcing it to select an existing node. Attention mass and null-node selection are learned features, not certificates of missing constraints.

Alignment operates on copies of node features, leaving the graph-only values unchanged. Source-aware losses can still update the shared graph-encoder parameters during training. Separate input views therefore do not imply separate training gradients.

Each event combines its IR embedding with embeddings for branch, IR type, compiler pass, execution status, construction mode, and producer. Recorded correspondences link events to aligned source-unit features. Two dependency-aware attention blocks process the run, using parameters separate from those of the paired event path. A $768 \to 256$ projection combines original-question, context, and answer/proof features into a condition vector. This vector and the alignment summary condition event processing.

Claim-conditioned attention pools the events. A $1024 \to 256$ fusion then combines the event pool, alignment summary, pooled source, and condition to produce an individual-claim logit. Individual-run scoring requires neither a comparison partner nor a reference graph.

Individual-claim scores are retrospective: they require a completed asserted claim and completed execution. A nonasserting or failed run receives no score, rather than a negative hallucination label. In the frozen held-out study, the optional heads for missing source units, node defects, individual divergence, and contribution have no targets, so their losses are masked. These unsupervised heads provide no validated localization or attribution result.

D.3 Paired comparison and construction metadata

The paired path compares the same feature types across runs. Equation 6 combines sums, absolute differences, products, and signed differences; each type uses a $1024 \to 512 \to 256$ projection with GELU and LayerNorm. The first three operations are symmetric under exchange of the runs, while signed differences preserve their order.

Four scalars supplement each feature comparison: cosine similarity, $\log(1+\mathrm{RMS}(v_R - v_L))$, $\log(1+\mathrm{mean}\,|v_R - v_L|)$, and a stabilized $\tanh$ of the log right/left norm ratio. Together, the program, query, source, question, and answer/proof views supply 20 values to a $20 \to 128 \to 256$ projection. The alignment-summary comparison is added to the mean paired-event state. The seven resulting streams then enter a $1792 \to 512 \to 256$ fusion.

Recorded events are aligned by registered boundary and invocation number. Repeated calls remain distinct, and unmatched events retain explicit left-only or right-only tokens. Two typed eight-head attention blocks use both runs' forward/reverse dependencies and the source/question/answer

changes relevant to each branch. A $256 \to 5$ head predicts the sensitivity class. Auxiliary heads predict the paired claim label, decision relevance, candidate change, and ordered-pair error; a nonlinear source/context gate conditions the relevance path.

Paired collapse/propagation heads use the restricted construction labels defined in Appendix E. The paired claim score is learned separately: it predicts whether either run contains an unsupported query claim and is not computed as the maximum or probabilistic union of the individual scores.

Construction modes. A construction mode describes how Policy Verifier produces a candidate artifact before the detector receives it:

Deterministic. Rule-based parsing and compilation construct the artifact without LLM generation.

Default. An LLM extracts or formalizes the input before compilation.

Guided. An LLM proposes refinements to a compiler baseline at intermediate and final stages; symbolic checks govern acceptance.

LLM-only. An LLM directly generates the requested artifact, bypassing parser/compiler construction.

Validation and execution status are recorded separately. Policy and query construction modes are recorded independently; a producer field identifies the component responsible for each event, such as a parser, compiler, or LLM. Separate categorical embeddings encode these fields in source-aware event processing, outside the graph-only view.

An *unknown* value means the setting was not recorded; it does not mean Default. All training and development records in the frozen study carry unknown mode and producer settings. The reported results therefore provide no comparison of detector performance across these four construction modes. Missing historical checkpoints also remain missing observations.

D.4 Targets, optimization, and initialization

Equation 7 first averages losses over available labels within a pair, then sums weighted terms and averages across pairs. All active term weights are one. Cross-entropy supervises sensitivity classes and candidate-execution outcomes. Binary cross-entropy (BCE) with logits supervises individual claims and the paired claim label. Relevance, candidate change, and ordered-pair error use BCE with positive weight $2/3$. Boundary BCE averages losses over available labels separately for collapse and propagation.

Losses for left and right execution targets are averaged within the execution term; losses for asserted claims are averaged within the claim term. Missing supervision contributes no loss. Applying a sigmoid produces bounded scores, but does not by itself calibrate them as probabilities.

The candidate-execution head predicts the program's four-state support/refutation outcome. Reference outcomes separately supply sensitivity and claim-support labels; they are never detector input features. The diagnostic outcome probe receives a detached embedding and uses its own optimizer, so its reference-outcome loss cannot update the graph encoder. No independent diagnostic-probe measurement or information-preservation benefit is reported. Freezing a readout's parameters while retaining gradients to its input, as in the optional objective below, is a different operation.

The frozen detector inherits compatible weights from an earlier detector fitted on the same training role and selected using the same development role. The execution readout and new modules are initialized separately. Training uses a fresh AdamW optimizer, seed 11, learning rate 10^{-4}, weight decay 0.01, effective batch size 32, gradient clipping at 1.0, and BF16 precision. The reporting model is fixed after 22 epochs, before held-out access. Batch partitioning retains 10,656 of 10,675 shuffled training pairs per epoch, giving 333 synchronized updates. The warm start and prior development inspection limit the interpretation of development performance.

The supplementary configuration records the computing environment. The pretrained backbone stays frozen; repeated-input predictions agree within 10^{-5} under deterministic evaluation. The

supplementary individual-supervision study instead initializes the extended detector afresh. Appendix D.8 specifies its added components, and Appendix E.7 specifies its data and loss.

D.5 Gradient through a fixed outcome readout

A fixed outcome readout illustrates how an information-preservation loss differs from detached probing. This analytical objective is not part of detector training. For a separately fitted readout $\rho_{\bar{\eta}}$ with fixed parameters $\bar{\eta}$, define

$$\mathcal{L}_I = -|\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} \log \rho_{\bar{\eta}}(Y_i^* \mid z_\phi(G_i, Q_i), Q_i),$$

where $\mathcal{I}$ indexes examples with complete reference labels and $\log$ denotes the natural logarithm for this training loss. Unlike detached diagnostic probing, this path retains the derivative with respect to the embedding. With $J_\phi z_\phi$ denoting its Jacobian,

$$\nabla_\phi \mathcal{L}_I = -|\mathcal{I}|^{-1} \sum_{i \in \mathcal{I}} J_\phi z_\phi(G_i, Q_i)^\top \, \nabla_z \log \rho_{\bar{\eta}}(Y_i^* \mid z, Q_i)\big|_{z = z_\phi(G_i, Q_i)}.$$

Only the representation parameters receive this gradient; programs, solver outputs, labels, and readout weights remain fixed. A readout used in this loss is part of the learning objective and cannot also supply independent evidence of information preservation. Such evidence requires a separately fitted diagnostic after the representation is fixed. The derivative establishes the gradient path, not an empirical benefit.

D.6 Finite-candidate generator objective

A separate analytical objective concerns an upstream generator that assigns probabilities to a fixed candidate pool. For candidates G_i, define $p_i = \mathrm{softmax}(\log p_\theta(G_i \mid X)/\tau_{\mathrm{gen}})$, where $\tau_{\mathrm{gen}} > 0$ is the temperature. Let c_i be fixed logical or information costs and $\bar{c} = \sum_i p_i c_i$ their expected value. Differentiating the candidate probabilities gives

$$\nabla_\theta \sum_i p_i c_i = \tau_{\mathrm{gen}}^{-1} \sum_i p_i (c_i - \bar{c}) \nabla_\theta \log p_\theta(G_i \mid X).$$

The candidate pool and costs stay fixed; gradients do not pass through decoding or logical evaluation. Central differences verify the derivative numerically. This analytical identity supplies no empirical result for generator training, candidate reranking, or the detector evaluated here.

D.7 Evidence for detection and replay

A detection finding must identify the source policy, original question, fixed context, candidate artifacts, and completed assertion. Independent reference interpretations and labels belong to evaluation and remain outside learned inference. A neural score expresses suspicion; a completed semantic check establishes a finding under its stated assumptions.

Tracing requires the observed processing stages, their dependencies, semantic outputs, source correspondences, and execution status. Repeated invocations remain distinct. Missing predecessors leave earlier origin unresolved, and an observed defect is not automatically a contributor to the final claim.

Replay additionally records the original and repaired artifacts, the source-based justification for the replacement, and the dependent stages that were repeated. Source, question, context, and external inputs remain fixed. The comparison follows the same proposition and separately measures changes in candidate justification and claim emission. Control interventions assess whether the effect is specific to the proposed repair. These requirements keep neural nominations, validated discrepancies, and intervention-supported contribution distinct.

D.8 Checkpoint transitions and policy–query interaction

The checkpoint extension lets the individual-claim predictor compare artifacts across observed processing stages. The base encoders retain four blocks, width 256, and eight heads. Node correspondences connect artifacts along recorded producer–consumer dependencies. Matching identifiers

supply candidate links; when identifiers change, unique operator-subtree or rule signatures provide a fallback. Ambiguous matches remain unresolved. A correspondence records a structural match, not a guarantee that meaning was preserved.

For each observed dependency, the model combines feature sums, absolute and signed differences, elementwise products, and the dependency type. It compares linked-node features where correspondence is available and pooled artifact states otherwise. Stage embeddings identify the observed boundary. Bidirectional attention also compares policy-program nodes directly with query nodes; its summary enters the individual-claim predictor.

These additions preserve the source-free program/query embeddings and candidate-execution readout. Individual-claim scoring still needs only one run, and missing compiler events remain missing. The extended detector has 25,108,216 trainable parameters, including inactive auxiliary heads.

Supervised checkpoint targets. Five heads predict source-relative divergence, a newly introduced local error, propagation of an incoming error, the earliest observed divergence, and repair-supported contribution. The controlled traces record graph, compiled-program, query, proof/execution, and answer boundaries. Missing predecessors leave earlier origin unresolved. Contribution labels mark membership in minimal effective repair sets within the declared search over single and joint repairs of size at most two. Membership establishes neither a unique cause nor that one member suffices alone. Labels and reference artifacts remain outside inference.

Each head applies layer normalization and a $256 \to 128 \to 1$ network to contextual event features, with GELU between linear layers. The five heads contain 167,685 parameters. One study freezes the detector and fits only these heads. A subsequent configuration adds a $768 \to 256$ adapter combining the event state, policy–query summary, and mean incoming-transition features, with a residual event-state connection. Its completed development evaluations are reported in Appendix E.8; source-span and node-level localization remain unevaluated.

Separating claim and localization learning. A further training configuration uses context contrasts: the candidate program, query, and asserted proposition stay fixed while one known fact changes whether the source supports the claim. The validated contrasts supply pointwise claim labels, not a contrastive loss or an inference partner. After deduplication, 12,984 additional fitting examples contain 7,892 supported and 5,092 unsupported claims. References and pair identities remain evaluator-only.

Claim BCE updates the detector trunk. Boundary losses update only their adapter and heads: event, policy–query, incoming-transition, and residual-path features are detached from the trunk. The objective is mean claim BCE plus 0.1 times the mean of the available per-head boundary BCEs. Claim and boundary parameters have separate gradient-norm clipping limits of one, so boundary gradients cannot rescale the claim update.

AdamW uses trunk learning rate 10^{-5}, head learning rate 10^{-4}, weight decay 0.01, effective batch 2,048, and seed 11; the text backbone remains frozen. This configuration starts from joint-training weights with a fresh optimizer. For reporting, one checkpoint is chosen retrospectively for its development F1–false-positive-rate trade-off at threshold 0.5, replacing maximum-AP selection. All claim and boundary metrics use that model. Data, gradient routing, and optimization change together, so the result cannot isolate the effect of any one change.

E Detection results and diagnostic analyses

This section separates held-out detection results from development measurements and the post-test capacity diagnostic. Supplementary studies assess individual supervision and checkpoint prediction on development data only. Claim metrics require completed assertions with independent labels. Controlled localization and repair results do not establish human-validated attribution.

E.1 Five labels and construction scope

For two supplied runs with the same question and context, write (y_L^*, y_R^*) for the independently established reference outcomes and $(\widehat{y}_L, \widehat{y}_R)$ for completed candidate executions. Define reference

change $d_{\text{ref}} = \mathbf{1}[y_L^* \neq y_R^*]$, candidate change $d_{\text{cand}} = \mathbf{1}[\widehat{y}_L \neq \widehat{y}_R]$, and ordered-pair error $e_{\text{ord}} = \mathbf{1}[(y_L^*, y_R^*) \neq (\widehat{y}_L, \widehat{y}_R)]$. The five mutually exclusive classes are

Condition	Pair class
$d_{\text{ref}} = 0, d_{\text{cand}} = 0$	invariant preserved
$d_{\text{ref}} = 1, d_{\text{cand}} = 1, e_{\text{ord}} = 0$	preserved change
$d_{\text{ref}} = 1, d_{\text{cand}} = 0$	lost distinction
$d_{\text{ref}} = 0, d_{\text{cand}} = 1$	spurious change
$d_{\text{ref}} = 1, d_{\text{cand}} = 1, e_{\text{ord}} = 1$	wrong direction

These classes describe the response to an edit. They do not determine whether each outcome is correct. For example, *invariant preserved* includes two identical but incorrect candidate outcomes; the ordered-pair error target records that error. *Wrong direction* means both pairs change but their ordered outcomes disagree. Reversing the reference outcomes realizes this class in the controlled construction, although its definition also covers other incorrect changes among multiple outcome values. Incomplete reference or candidate outcomes receive no definitive sensitivity label.

For each validated mutation witness, let P be the original policy and P' its changed policy, with different reference outcomes for the fixed question and context. The controlled builder forms five reference-pair / candidate-pair combinations: $(P, P)/(P, P)$, $(P, P')/(P, P')$, $(P, P')/(P, P)$, $(P, P)/(P, P')$, and $(P, P')/(P', P)$. These instantiate the labels in table order.

Controlled alternate renderings supply the source text, and the bounded harness receives the specified executable candidates. This construction tests detection on supplied artifacts rather than natural text-to-program decoding. Each mutation witness generates five related pairs. The 2,135 training and 535 development witness groups belong to 600 and 150 families, respectively.

The paired claim label is one when at least one run asserts a query claim that its reference classifies as a hallucination. Classification retains the assertion status and whether conflict is acknowledged. This pair-level label does not identify the responsible side, an erroneous text span, or a contributing defect. Development contains 1,159 positive and 1,516 negative pairs.

Only lost-distinction constructions receive a collapse target at the first recorded policy-graph boundary and propagation targets at compilation, execution, and emission. These are construction labels, not validated origins. On 54,225 original-test boundary labels, the learned collapse score has AP 30.77% but zero recall at threshold 0.5. A constant first-boundary predictor has 100% recall, 20% precision, and 33.33% F1 because of the construction. This control prevents interpreting boundary accuracy as causal localization.

E.2 Development and held-out populations

The development population contains 2,675 pairs and 5,350 side occurrences. Deduplicating complete inference inputs yields 2,994 distinct inputs, of which 1,338 are evaluable assertions: 902 unsupported and 436 supported. At threshold 0.5 the detector has 882 true positives, five false positives, 20 false negatives, and 431 true negatives. Sensitivity accuracy and macro-F1 are both 99.85%. These repeatedly inspected development measurements are descriptive; the held-out evaluations below provide the generalization evidence.

The original test contains 600 families and 10,845 pairs, with 2,169 pairs in each sensitivity class. Its 21,690 trace occurrences include 11,181 asserted occurrences. Deduplication yields 11,544 distinct inputs, of which 4,635 have evaluable assertions: 3,034 positive and 1,601 negative. The remaining 6,909 inputs are outside the binary claim denominator. All attempted pairs execute successfully, and every evaluable claim receives a score.

The standalone evaluation contains 360 families and 10,080 unique runs, including 4,014 asserted occurrences and 6,066 nonasserting runs. There are 3,974 definitive claim labels: 2,604 unsupported and 1,370 supported assertions. Forty asserted runs have no definitive claim label. Across all attempts, 720 recorded executions fail and retain unavailable labels. Separately, the detector's graph budget rejects 2,626 input records, including 1,004 evaluable claims. These input failures must not be counted as supported claims or deliberate answer abstention.

Table 3: Raw claim confusion counts. Unscored positives remain in the denominator of all-evaluable recall, but not scored-claim precision, recall, F1 or AP. The capacity row reuses the same test population after a post-test input-budget change.

Population	TP	FP	FN	TN	Unscored +	Recall (all)
Original test	2,366	425	668	1,176	0	77.98
Standalone test	526	468	1,442	534	636	20.20
Capacity diagnostic	587	534	2,017	836	0	22.54

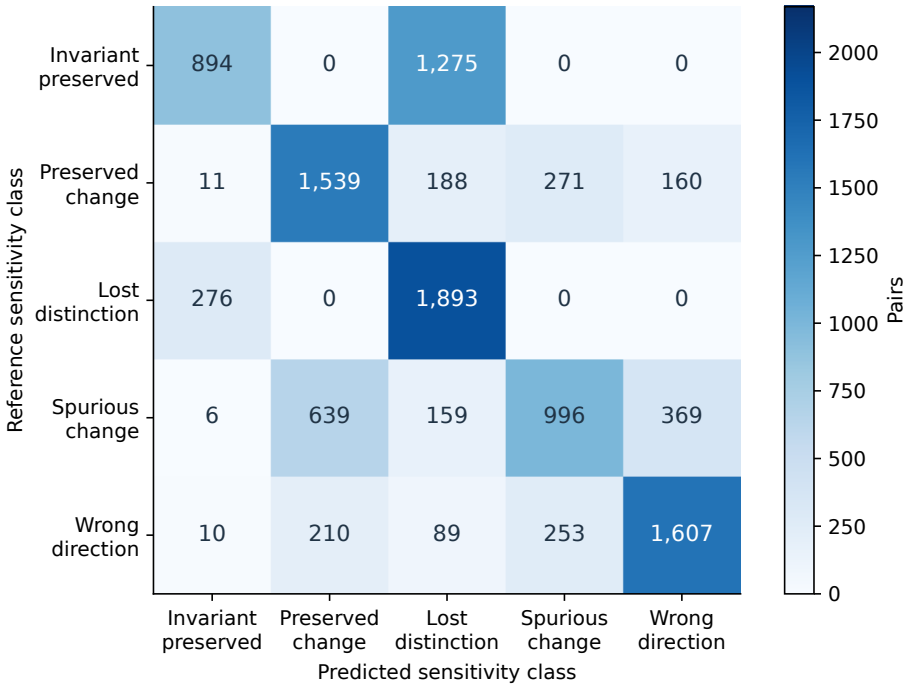


Figure 6: Original-test five-class sensitivity confusion matrix. Rows are reference classes and columns are predictions; every row contains 2,169 pairs. Counts, rather than normalized colors alone, expose the dominant invariant-to-lost confusion.

E.3 Sensitivity errors and claim generalization

Original-test sensitivity accuracy is 63.89% (95% family-bootstrap interval 62.44–65.38%), with macro-F1 62.98% (61.33–64.68%). Figure 6 shows the complete confusion matrix. Lost distinctions are often recognized, but invariant pairs are frequently assigned that label. Sensitivity classification therefore cannot substitute for checking the source support of an individual assertion.

At threshold 0.5, original-test individual-claim F1 is 81.24% and FPR is 26.55%; AP is 91.88%. Their respective 95% intervals are 80.13–82.31%, 24.53–28.61%, and 91.26–92.52%. The constant-score AP baseline equals positive prevalence, $3{,}034/4{,}635 = 65.46\%$. Ranking exceeds this baseline, but the default threshold still produces substantial false alarms.

On scored standalone claims, F1 is 35.52% (31.14–40.17%), AP is 56.56% (53.22–60.45%), and FPR is 46.71% (40.62–52.47%). Recall across all evaluable positives, including unscored positives as misses, is 20.20% (16.72–23.99%). All intervals use 2,000 family-clustered resamples; they do not estimate variation across training seeds.

A structural audit removes names while retaining operator roles, scope, time, and shared atoms. It finds 45 original-test signatures, none matching the 48 pooled prior signatures. The standalone candidate audit finds 1,800 distinct signatures, also without a prior match. Because it inspects candidate artifacts, the latter audit cannot establish independence of reference structures. These fingerprints

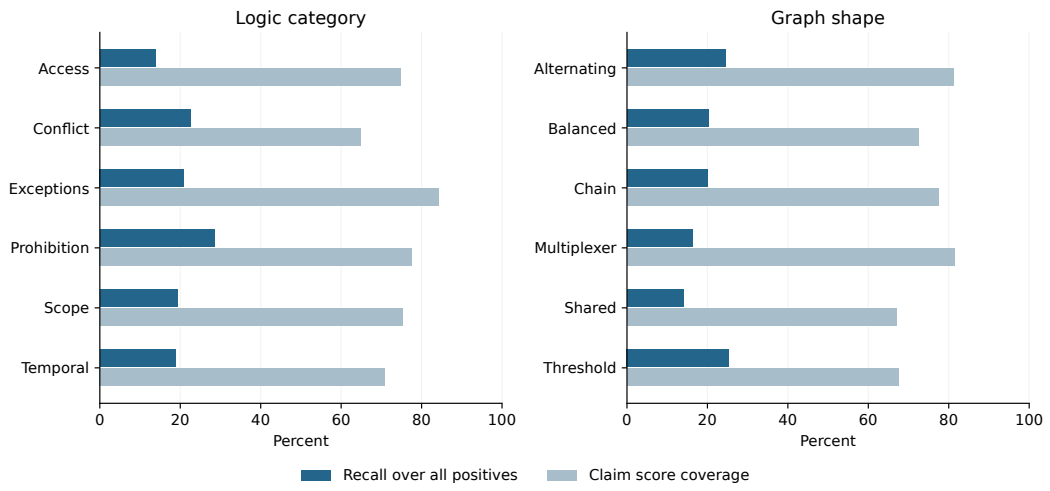


Figure 7: Standalone claim performance by logic category and graph shape at threshold 0.5. Recall uses all evaluable positive claims, including unscored positives as misses. Coverage uses all evaluable claims. Each logic category and each graph shape contains 60 families and 1,680 attempted cases; label counts vary.

Table 4: Operating-point transfer. All values except threshold are percentages. Standalone recall and FPR here condition on the 2,970 scored claims. Development FPR targets are constraints on development observations, not guarantees. Temperature scaling preserves ranking and thus AP.

		Original test			Standalone test		
Setting	Threshold	F1	R	FPR	F1	R	FPR
Raw	0.50000	81.24	77.98	26.55	35.52	26.73	46.71
Development max-F1	0.17344	81.11	79.17	30.42	38.61	30.39	53.09
Development 1% FPR	0.48246	81.27	78.08	26.67	35.78	26.98	46.81
Development 5% FPR	0.17344	81.11	79.17	30.42	38.61	30.39	53.09

detect specific structural overlaps; they do not certify semantic equivalence or independence. Together with the development/test gap, they motivate tests of structural transfer without identifying a unique cause of failure.

E.4 Calibration and selective answer filtering

Temperature scaling minimizes negative log likelihood on 75 reserved calibration families containing 669 distinct evaluable assertions. The fitted temperature is $T = 1.05072$, reducing loss on that calibration population from 0.035648 to 0.035547. A separate 75-family threshold role selects 0.17344 for maximum F1, breaking ties toward the higher threshold. The empirical 1% and 5% FPR targets select temperature-scaled thresholds 0.48246 and 0.17344. All settings are frozen before test access.

Raw Brier score is 0.2233 on original test and 0.6151 on scored standalone claims; temperature scaling changes these to 0.2225 and 0.6131. The development-selected maximum-F1 threshold yields F1 81.11% and 38.61%, respectively. The development 1% FPR setting attains test FPR 26.67% and 46.81%. The large reliability and transfer errors show why a good development operating point is insufficient.

The deterministic readiness check provides an access-matched rejection heuristic, not a proof of source support. On the original evaluable population it rejects no claims (recall zero). On the same 2,970 scored standalone claims used by the neural comparison, it has 30 true positives, eight false positives, 1,938 false negatives, and 994 true negatives: precision 78.95%, recall 1.52%, and F1 2.99%. Its coverage is matched to the neural scored population for this comparison, rather than

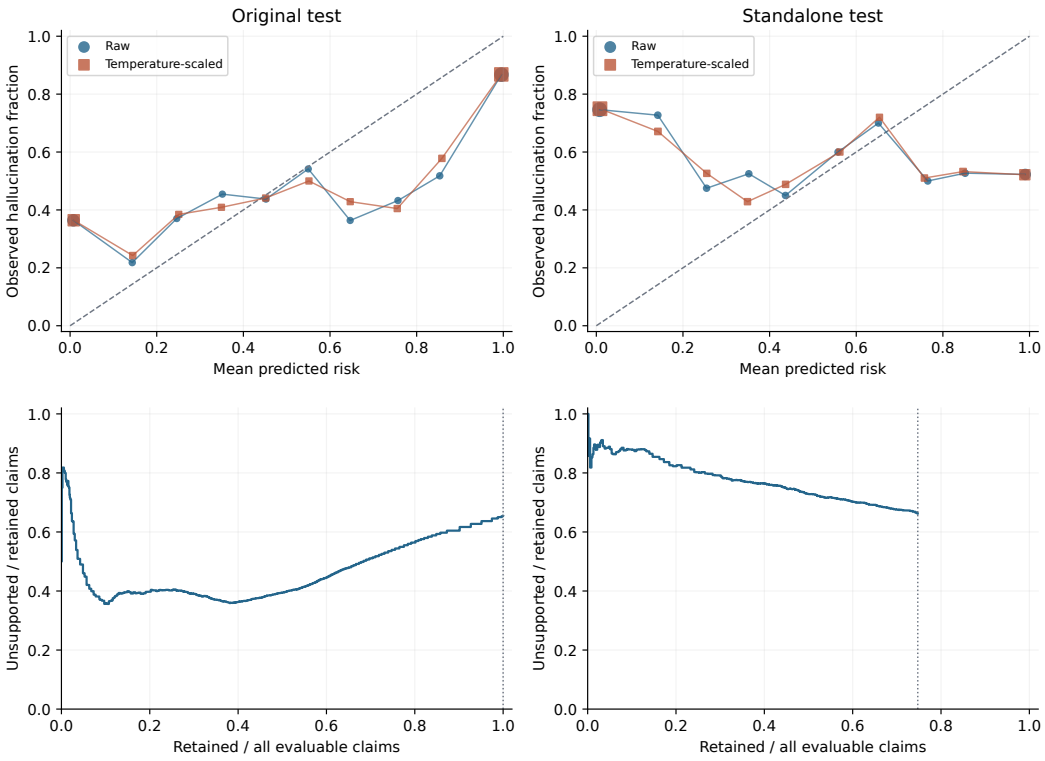


Figure 8: Held-out score reliability and answer-filtering risk–coverage. Top: observed hallucination incidence versus mean score in ten equal-width bins, with marker area proportional to bin count. Bottom: retain claims in ascending risk-score order, including tied scores together; risk is the unsupported fraction among retained claims. Coverage is relative to all evaluable claims, so standalone coverage ends at 74.74%. Zero-coverage risk is undefined.

interpreted as an intrinsic capacity limit of the readiness checker. No matched learned-architecture comparison is reported.

E.5 Input capacity

Every standalone input-budget failure is caused by an artifact exceeding the 640-node IR limit; no constraint is silently truncated. Original-test artifacts contain at most 601 nodes and standalone artifacts at most 909. The node limit also sets the scale of array-index features. Simply increasing that scale would alter representations even for previously admitted inputs.

The capacity diagnostic raises the input limit to 2,048 nodes while preserving the original position scale and all learned weights. On 24 sampled fitting/development pairs already within the original limit, features and predictions remain identical. This consistency check does not establish accuracy on larger inputs.

All 2,626 records rejected by the input budget are re-evaluated without label-based selection; previously scored predictions are retained. Scores are recovered for all 1,004 unscored evaluable claims, including 636 positives. Only 61 additional positives exceed threshold 0.5. Recall across all evaluable positives rises from 20.20% to 22.54%. Complete-population AP is 56.44%, below its 65.53% constant-score baseline. F1 is 31.52%, now measured on a larger scored population. Recovering coverage therefore leaves substantial detection error. This diagnostic reuses an inspected test population and is reported separately from the frozen evaluation.

E.6 CONTROLLED TERMINAL-ARTIFACT INTERVENTIONS

The replay evaluation holds the original source, proposition, and context fixed while replacing compiled-program, query, or answer components with independently validated alternatives. It evaluates all eight subsets, including the null intervention. Of 10,080 attempted cases, 8,640 complete and 1,440 remain unresolved: 720 lack checkpoints and 720 record failed execution. The 69,120 completed repair configurations preserve source support. Null and reverse controls pass in all 8,640 completed cases.

A contributing component set is found in 2,050 cases. This count measures the effects of supplied repairs on terminal artifacts; it does not measure neural nomination accuracy. The repairs need not identify a minimal source edit, and the earlier origin of a compiler defect remains unresolved. This deterministic evaluation does not measure alternative faithful repairs or stochastic replay. Cases with an upstream correction still require claim-level checks, since correcting one defect need not make every emitted assertion supported.

E.7 SUPPLEMENTARY DEVELOPMENT ANALYSIS

The extended individual-claim detector in Appendix D.8 uses 3,600 training families (100,800 cases) and 720 development families (20,160 cases), spanning six logic categories and six graph shapes. Each family supplies two renderings and fourteen variants. Separate calibration and threshold pools contain 360 families each. Name-insensitive signatures exclude detected overlap across roles and with the original and standalone registries, without certifying semantic independence.

Detector parameters are initialized afresh and the text backbone remains frozen. The objective is claim BCE plus 0.2 times candidate-execution cross-entropy, each normalized over available labels. AdamW uses learning rate 10^{-4}, weight decay 0.01, effective batch 2,048, clipping at 1.0, and seed 11. Paired and localization losses are inactive.

One checkpoint is selected by maximum development claim AP, with ties broken by lower claim log loss and then earlier checkpoint. All reported metrics come from that checkpoint. Its evaluation scores all 7,730 evaluable assertions among 20,160 inputs without inference failures. At threshold 0.5, there are 3,616 true positives, 450 false positives, 1,506 false negatives, and 2,158 true negatives. AP is 92.49%, precision 88.93%, recall 70.60%, F1 78.71%, and FPR 17.25%; Brier score is 0.1551 and log loss 0.4517. Ranking exceeds the 66.26% constant-score AP baseline.

These measurements use the development population that selects the checkpoint. Differences in data, initialization, and training budgets prevent a matched architectural comparison. This configuration has no reported held-out or calibrated result.

E.8 SUPERVISED CHECKPOINT DEVELOPMENT STUDY

The checkpoint corpus uses 3,600 fitting families and 720 development families, each with two renderings and twelve trace constructions. The constructions include equivalent rewrites; defects in graphs, compiled programs, queries, proofs, or answers; joint defects; upstream corrections; missing graph observations; and failed execution. There are 86,400 fitting attempts and 17,280 development attempts. Deliberate execution failures leave 7,200 fitting and 1,440 development inputs unavailable. The remaining 15,840 development traces admit feature extraction. These are controlled constructions rather than retrospectively annotated production runs.

The frozen-detector study trains the five heads for 22 epochs and selects the checkpoint with lowest mean development head log loss. Table 5 reports its boundary predictions alongside the separate configuration that prioritizes claim learning. Every head retains its own denominator because different semantic checks can be unresolved. The constant-boundary baseline marks the boundary most frequently positive in fitting data. It provides a control for predictable construction locations.

With the detector frozen and checkpoint heads trained, divergence and propagation AP exceed the constant-boundary baseline. However, four heads predict no positives at threshold 0.5. Earliest-divergence AP is 22.09% (95% family-bootstrap interval 21.20–23.05%), and contribution AP is 22.90% (21.86–23.99%). All five heads have zero exact-set accuracy among runs with a nonempty

Table 5: Checkpoint predictions on 720 development families. AP and F1 are percentages; F1 uses threshold 0.5. Each row counts evaluable boundaries, not independent runs. All metrics within a configuration use its selected checkpoint. The two configurations have different training procedures and are not a matched ablation.

		Constant	Frozen detector		Claim-prioritized	
Target	Labels	AP	AP	F1	AP	F1
Divergence	77,642	29.88	48.46	36.88	64.21	43.66
Local error	76,288	15.16	22.54	0.00	35.60	0.00
Propagation	77,642	20.72	45.48	0.00	69.04	45.39
Earliest divergence	77,642	15.45	22.09	0.00	40.20	11.83
Contribution	38,997	15.45	22.90	0.00	61.84	27.95

positive set. The intervals use 2,000 family resamples conditional on this fitted model; they do not measure variation across seeds.

Defects and supported claims. The trace population contains 7,993 evaluable assertions, including 5,344 unsupported and 2,649 supported claims. It differs from the 7,730-claim population above. A matched audit on these same traces finds 1,180 false positives already present in the initializing claim detector: all 295 supported claims in each of four constructions with graph, compiler, joint graph/compiler defects, or a missing graph observation are flagged. A defect can therefore be recognizable while the current assertion remains supported. Joint training also shifts scores: faithful-case FPR rises from 23.73% to 49.15%. These observations motivate context contrasts and separate gradient routing, but do not identify loss interference as the cause.

Claim-prioritized development evaluation. Using the fitting-only context contrasts and gradient routing in Appendix D.8, the reported checkpoint scores all 7,993 evaluable claims. At threshold 0.5, it yields 4,015 true positives, 670 false positives, 1,329 false negatives, and 1,979 true negatives: precision 85.70%, recall 75.13%, F1 80.07%, AP 89.99%, and FPR 25.29%. Brier score is 0.1814 and log loss 0.5317. The operating point retains substantial false alarms and missed unsupported claims. These measurements cover the complete declared development population; this configuration has no reported test or calibration result.

All checkpoint metrics use this same selected model (Table 5). Local-error recall is zero. Earliest-divergence recall is 6.44%, with exact-set accuracy 6.93% among 9,016 positive-set runs. Contribution recall is 16.24%, with exact-set accuracy 18.60% among 4,671 positive-set runs. Contribution predictions have 869 true positives and 0 false positives, but miss 4,481 positive boundaries. Their 100.00% precision therefore accompanies low recall. Retrospective development selection, differing optimization budgets, and combined interventions prevent a matched architectural or causal improvement claim. Neural scores remain nominations to check against source-grounded repair and replay.

E.9 Optimization and measurement scope

All 333 updates in the frozen detector's final reported training epoch require clipping (pre-clipping norm: mean 16,815.4; maximum 1,788,317.4). Decreasing loss therefore does not establish stable optimization; gradient magnitude is distinct from semantic sensitivity. The held-out results concern one fitted configuration and seed, without individual-localization supervision. The supplementary checkpoint studies also use one seed and controlled labels. Construction modes remain unrecorded; no study establishes mode-specific performance or natural-policy robustness.

F Attribution procedure and evaluation protocol

This section defines the attribution procedure, documents the controlled corpora, and specifies reporting units. The measured interventions have narrower scopes: held-out replay repairs terminal artifacts (Appendix E), while checkpoint development uses a bounded family of repairs (Appendix E.8). The general procedure also describes controls that have not all been evaluated.

F.1 BOUNDARY ATTRIBUTION ALGORITHM

Attribution starts from a completed claim (a, u), holding its source, original question, context, and reference label fixed. If the claim interpretation or label is unresolved, local defects and neural scores remain reportable, but claim attribution remains unresolved. Inspection follows the recorded dependencies backward from emission through both policy and query branches. Each boundary is checked for preservation of its inputs and agreement with the validated source and question. Unavailable comparisons remain explicit.

An observed divergence requires a completed semantic check that establishes a discrepancy; a syntax error alone does not establish one. The earliest divergences are those with no earlier established divergence along their recorded dependencies. Different branches can therefore yield a set of earliest events. A newly introduced error additionally requires faithful inputs followed by an unfaithful output under the relevant contract. Otherwise the discrepancy may have been inherited. Propagation records persistence of that discrepancy; a correction ends it unless it is later reintroduced. Missing predecessors leave earlier origin unresolved.

For each suspect component on these dependencies, an intervention supplies a replacement independently validated against the source. Replay keeps the reference fixed and compares candidate proof support $J(a)$ and claim emission $M(a)$. A contribution finding requires both correction of the intended discrepancy and a change in justification or emission. These two effects are recorded separately. Irrelevant edits, alternative faithful repairs, and reverse mutations are controls for specificity. Editing a candidate cannot change the source label for the original claim; any different unsupported claim emitted after repair is recorded separately.

With at most four independently validated suspect defects, the general protocol tests every permitted nonempty repair subset. A successful set is inclusion-minimal when none of its proper subsets succeeds. Larger or incomplete searches report successful tested sets without claiming minimality. Checkpoint supervision uses the narrower repair family of size at most two. In either setting, minimality is relative to the permitted replacements and fixed context. Redundant routes can require joint repair, while cancelling defects can make one repair expose another error. Contribution may therefore involve several checkpoints or remain unresolved.

Replay fixes external inputs, clocks, and tool results. The reported replay is deterministic and compares candidate justification and emission of the same proposition before and after repair. Stochastic replay would additionally require a declared coupling of stage-specific random draws; no stochastic contribution estimate is reported.

F.2 CORPUS INVENTORY, PROVENANCE, AND LIMITATIONS

Table 6 contrasts the two populations used for held-out evaluation. Both use generated bounded programs and controlled-language renderings. Candidate artifacts are supplied by construction, so these experiments assess detection of controlled failures rather than the frequency of errors in unconstrained text-to-program generation.

The standalone shapes vary how conditions compose: balanced trees combine branches, shared subexpressions reuse dependencies, threshold conditions require at least a specified number of facts, conditional selection chooses between branches, and alternating structures mix conjunction, disjunction, and negation. Its fourteen variants each appear in two renderings, giving 28 attempted runs per family. Faithful and equivalent cases distinguish harmless variation from defects; failed execution and missing observations remain explicit. The selected 360 families therefore yield $360 \times 28 = 10{,}080$ attempts, not that many independent policies.

Labels follow the original source and asserted proposition under the fixed context. An injected defect does not automatically receive a positive hallucination label: a dropped condition can be irrelevant to a particular supported claim. Conversely, successful execution cannot establish source support. Reference programs, mutation tags, and validation outcomes supply evaluator-side supervision and do not enter ordinary detector inference. Nonasserting runs and unresolved checks are retained outside the definitive binary claim denominator.

Table 6: Construction and evaluation units of the two held-out corpora. Related examples remain grouped by policy family. Counts distinguish paired sensitivity examples, individual inputs, and assertions with definitive claim labels.

Property	Original controlled corpus	Standalone benchmark
Logic coverage	Access, exceptions/conflict, bounded obligations	Access, prohibition, exceptions, conflict, temporal rules, scope
Graph structure	Fitting uses conjunction/disjunction chains; test adds mixed branching and negation	Chain, balanced tree, shared subexpressions, threshold conditions, conditional selection, alternating operators
Family unit	Two renderings, eight question/context cases, witnessed policy mutations	Two renderings and fourteen candidate-run variants; each variant uses a selected question/context case
Candidate changes	Dropped conditions; operator, scope, modality, deadline and exception edits; five pair classes per mutation witness	Faithful/equivalent controls; program, query and answer defects; conflict, upstream correction, joint defects, missing checkpoints and execution failure
Held-out selection	All 600 original test families	Ten families per logic/shape combination, selected by fixed hash order; 360 total
Evaluation units	10,845 pairs; 11,544 distinct individual inputs; 4,635 evaluable assertions	10,080 individual runs; 3,974 evaluable assertions; no constructed comparison partner

Table 7: Original-corpus family and surface–case allocation. Within these partitions, the frozen detector uses 600 fitting and 150 selection families. Calibration and threshold selection use separate development roles. Held-out evaluation uses the complete test partition.

Partition	Families	Surface–cases	Registered roles
Training	3,000	48,000	1,800 representation; 600 detector; 300 surrogate; 300 diagnostic
Development	300	4,800	150 selection; 75 calibration; 75 threshold
Locked test	600	9,600	Reserved structural compositions

Materialized controlled data. Table 7 partitions the original 3,900 policy families into 3,000 training, 300 development, and 600 locked-test families. Access, exceptions/conflict, and temporal obligations each contribute 1,300 families. Two controlled-language renderings and eight question/context cases per family yield 62,400 surface–case combinations. These are 16 observations per family rather than independent policies. Admission checks require agreement between independent Python and Z3 evaluators and a separate parse of each rendering. No families were quarantined; the language renderings were not validated by human reviewers.

The 3,300 non-test families contain 11,744 witnessed mutations but only 48 atom-renamed serialized program signatures, all recurring across fitting/development roles. Family isolation therefore does not imply structural independence. The corpus supports controlled composition tests, with limited structural and language diversity.

Construction uses seed 11, unique family identifiers, and fixed role assignments. Each role balances access, exceptions/conflict, and temporal obligations.

Record unit. A family is the unit of split assignment and dependence. Its record contains two controlled-language sources, a typed reference program, eight query/context cases, accepted mutations and their witnesses, rejected proposals, and role/stratum/template identifiers. Programs include Boolean nodes, normative rules, bindings, time intervals, deadlines, and separate priority/defeat relations. Context supplies fact observations, authenticated actions, time, and completeness.

Split IDs, references, and mutation tags are evaluator metadata and enter neither graph-only features nor ordinary monitoring. Claim evaluation additionally requires the emitted assertion and recorded trace; attribution requires validated replay evidence. Policy-family membership alone supplies none of these labels.

Coverage. The generator independently varies recorded rule counts $\{4, 8, 16\}$ and relevant construction depths $\{1, 2, 4\}$. Its bounded core uses employee/access bindings, record/archive resources, Boolean fact atoms, horizon 16, and specified trigger/deadline cases. Non-test programs use conjunction and disjunction chains; test programs use mixed branching and negated compositions. They do not establish unrestricted deontic/temporal coverage, broad domain transfer, or complete structural independence.

Admission and source checks. The exhaustive Python interpreter and independently implemented Z3 encoding must agree before a family is admitted. A separate controlled-text parser checks both source renderings on the eight declared cases. This restricted round trip checks the controlled language on those cases; it neither validates arbitrary language nor replaces human review. Agreement between the evaluators remains evidence on the declared cases, rather than a proof of evaluator correctness.

Table 8: Mutation inventory in the 3,300 non-test families. Each proposal is retained only with a distinguishing witness among the declared cases. Counts here describe the non-test inventory only.

Mutation	Witnessed	Rejected proposal
Dropped condition	3,300	0
Operator substitution	3,119	181
Scope change	1,836	364
Modality substitution	1,287	364
Deadline change	734	366
Exception error	1,468	181
Total	11,744	1,456

Of these families, 2,570 have four accepted mutations, 185 have three, 364 have two, and 181 have one. A rejected proposal is not proved globally equivalent merely because no declared case distinguishes it. A witnessed program defect also need not affect the current emitted claim.

Independence and effective diversity. The non-test inventory contains 6,600 source renderings and 26,400 question/context records, yielding 52,800 surface–case combinations. Including test gives 62,400 combinations. These expanded observations remain dependent within families.

The non-test audit replaces fact-atom names by first-occurrence indices while retaining serialized program structure. It yields 48 signatures shared across fitting/development roles. This normalization is not a complete graph-isomorphism or semantic-equivalence test. Separate family identifiers prevent direct family reuse but cannot prevent template memorization. The audit in Appendix E separately checks overlap with evaluated populations. Neither audit establishes broad domain generalization, and family-clustered intervals quantify only family-level sampling variation.

Natural policies and dataset governance. The reported evaluations use controlled policies. Evaluation on human-authored policies would require source provenance, permitted use, and document-level grouping to prevent leakage. Ordinary extraction or access-decision labels cannot substitute for independent support judgments about the actual emitted claims, and do not identify contributing checkpoints. No natural-policy or human-validated attribution result is claimed.

F.3 Metrics and denominators

Attempted runs, emitted assertions, labelled claims, and scored claims are distinct counting units. Let N_A count attempted source/question runs. After deduplicating inference inputs, let N_E count emitted assertions, $N_L \leq N_E$ those with definitive independent labels, and $N_S \leq N_L$ those receiving a detector score.

Claim precision, recall, F1, AP, FPR, and Brier score use the scored, labelled subset. Label coverage is N_L/N_E and score coverage is N_S/N_L. Recall over all evaluable positives also counts unscored unsupported claims as misses.

Table 9: Required controls and intended interpretation.

Case	Required behavior
Equivalent wording / graph rewrite	Same bounded behavior, no defect from structure alone
Condition dropped, unrelated query	Defect may exist; supported assertion has label zero
Wrong operator, identifying structure	Recoverable outcomes do not certify faithful execution
Wrong query modality or time	Localize query defects separately; attribution requires replay
Conflict, pending obligation, incomplete trace	Keep conflict/status/uncertainty distinct
Joint or cancelling defects	Set-valued or unresolved contribution
Earlier error corrected downstream	Do not label it a final contributor without replay effect
Missing checkpoint / timeout / invalid graph	Retain unresolved checks and origin; do not infer support from missing evidence

Inventory counts retain nonasserting runs, invalid artifacts, unavailable labels, and execution failures. Occurrence counts are reported separately from deduplicated counts. Undefined denominators remain undefined; a population without positive labels has no defined positive-class recall. Claims from the same policy family remain clustered.

Detector abstention declines a verdict on an emitted claim; generator abstention emits no policy commitment. Detector selectivity reports the accepted fraction of N_L and classification error among accepted labelled claims. Answer filtering reports the retained fraction of N_L and the unsupported fraction among those retained. Risk at zero coverage is undefined.

Generator answer coverage divides asserting run occurrences by attempted occurrences N_A, keeping both counts in the same units. Acceptance and unresolved shares over all N_E reveal selection effects involving unlabelled claims.

Localization accuracy requires independent labels for source spans, graph locations, or checkpoint sets. The held-out model has no individual-localization labels; its paired-boundary scores use restricted construction labels. The separate supervised checkpoint study reports boundary labels and positive-set accuracy, without source-span or node-localization results. Controlled replay instead reports attempts, completed comparisons, contributing component sets, and unresolved origins. Changes in justification and claim emission remain separate, and unavailable repairs remain in the attempted-case count.

Failure-inclusive execution disagreement counts candidate failures among cases with complete reference outcomes. Disagreement among completed executions uses a separate denominator. Unsupported-permission mass, unsupported denial, and missed permission have distinct definitions for completed comparisons (Appendix C.4). These semantic quantities do not replace claim-detection metrics.

Reported 95% confidence intervals use 2,000 policy-family bootstrap resamples, retaining dependent examples within each sampled family. They describe sampling uncertainty conditional on the fitted model. One training seed supplies no estimate of variation across seeds, and bootstrap intervals do not substitute for human validation.

G Evaluation integrity and access regimes

Before held-out access, the evaluation fixes the detector, input view, metrics, calibration procedure, and operating points. Detector fitting excludes reserved calibration, threshold-selection, and test families. Predictions are recorded before joining labels for scoring. Unavailable labels and scores retain explicit status.

Exposure and selection. Recorded fitting and development inspection for the frozen detector involve 600 fitting and 150 selection families. Their identifiers do not overlap calibration, threshold, or test roles. This role audit does not address backbone-pretraining contamination. Benchmark manifest summaries and structural metadata were inspected during protocol design; individual test labels were excluded from model inputs and subset selection.

The original test uses all 600 test families. The standalone subset uses a fixed hash ordering to select ten families in each of 36 semantic-category/graph-shape strata, independently of labels and model scores. The remaining 8,649 standalone test families contribute no reported result.

Input and score boundaries. Each of the 10,080 standalone cases supplies its original source, question, context, candidate program and query, proof, assertion, execution state, and available checkpoints. Independent checks reproduce source and claim-label records on the 9,360 completed executions, including unresolved label status; 720 injected failures retain unavailable labels. No synthetic comparison partner or inferred historical event is supplied. Semantic-record features exclude source text, provenance, reference programs, and labels. Source-aware monitoring separately receives the permitted original-source view.

Operating points and uncertainty. Calibration fitting and threshold selection use separate development roles (Appendix E). Claim metrics use their stated scored populations, while coverage and all-evaluable recall retain missing scores. Sensitivity metrics count pairs, not individual claims. Confidence intervals use 2,000 policy-family resamples and describe uncertainty conditional on the fitted detector. They do not capture variation across training seeds or uncertainty introduced by prior development choices.

Why automatic detection remains difficult. A verifier establishes consequences of the program it receives. Determining whether the source authorizes those consequences also requires interpreting conditions, scope, exceptions, the original question, and available observations. A program can omit a dependency yet remain internally consistent. An equivalent rewrite can change structure while preserving meaning. Distinguishing these cases requires source information or additional assumptions: the program alone need not reveal what it omitted.

Supervised representations provide a ranking signal on the original controlled test, where AP exceeds positive prevalence. The complete-population standalone capacity diagnostic falls below that baseline. A useful ranking on one population therefore does not establish a reliable decision threshold or transfer to another.

With a trusted formal reference and complete bounded semantics, symbolic comparison can establish support directly. The empirical challenge is to detect unsupported claims without that reference at inference, including on unfamiliar policy structures. The reported transfer failures establish neither computational hardness nor impossibility.

Comparisons and interpretation. Readiness rejection uses only candidate inputs and supplies a limited deterministic baseline. Trusted-reference symbolic comparison has additional access: its outcomes supply evaluation labels, rather than an access-matched detector baseline. There are no completed matched comparisons isolating graph versus serialized structure, source alignment, or checkpoint access. Accordingly, the results do not establish an architectural improvement. Construction labels for pair boundaries and terminal-artifact replay effects also do not establish human-validated neural localization.

The capacity check follows test inspection and is reported separately from the frozen evaluation. Supplementary models use their own fitting and development populations (Appendices E.7 and E.8); previously inspected tests provide no untouched-test evidence for them. Development errors informed the context contrasts and gradient routing, so these results remain development diagnostics.

H Related-work comparison and scope of evidence

The comparison below places policy sensitivity, claim support, and artifact repair alongside related work. It compares objectives and observable inputs, without treating differences in task or mechanism as evidence of superior performance.

Source support and faithful formalization. FActScore measures source support for atomic facts in generated text (Min et al., 2023). RAGTruth provides manually annotated retrieval-augmented responses, including word-level hallucination spans (Niu et al., 2024). SelfCheckGPT uses agreement with sampled responses, while Chain-of-Verification uses verification questions to revise an

Table 10: Related objectives and their connection to paired policy verification. Entries describe tasks and mechanisms, not comparative performance.

Prior direction	Established objective or mechanism	Connection to this paper
Factual support (Min et al., 2023; Niu et al., 2024)	Atomic-fact support measurement and annotated hallucination spans	Source-policy support for the proposition actually asserted
Consistency and verification (Manakul et al., 2023; Dhuliawala et al., 2023)	Sampled-answer consistency and verification questions	Inspect the program and trace supporting an answer
Formal reasoning (Tafjord et al., 2020; Sanyal et al., 2022; Kamoi et al., 2025)	Proof construction, modular inference, and formal step labels	Logic-derived labels for changes across supplied verifier runs
Policy autoformalization (Mondl et al., 2026)	Generator–critic translation into Cedar policies	Learn representations of programs and traces supplied by Policy Verifier
Behavioral testing (Ribeiro et al., 2020)	Invariance and directional expectations under input edits	Compare required policy outcomes with executed outcome pairs
Graph representation learning (Li et al., 2019; Rampášek et al., 2022)	Learned graph similarity and hybrid message passing/attention	Shared encoders followed by source-aware, query-conditioned pair features
Semantic entropy and probes (Farquhar et al., 2024; Kossen et al., 2024)	Uncertainty across answer meanings; hidden-state prediction of that uncertainty	Distinguish answer uncertainty from graph-conditioned outcome recoverability
Information probing / MDL (Pimentel et al., 2020; Voita & Titov, 2020)	Recoverable label information and predictive effort	Analyze recoverability separately from execution and claim support
Graph intervention (Kong et al., 2026)	Counterfactual refinement of graphs derived from LLM internals	Define source-grounded artifact repair and downstream replay

answer (Manakul et al., 2023; Dhuliawala et al., 2023). These methods address evaluation, detection, and revision through different evidence. Here, the target is an asserted policy proposition: a proof against an incorrectly translated program does not establish source support.

ProofWriter's proof generation, FaiRR's modular inference, and FoVer's formally derived step labels motivate structured reasoning and supervision (Tafjord et al., 2020; Sanyal et al., 2022; Kamoi et al., 2025). Policy autoformalization combines deterministic checks with an LLM critic to assess source alignment (Mondl et al., 2026). That pipeline constructs and critiques translations. Our detector instead receives candidate runs and learns individual-claim support and paired semantic sensitivity.

Behavioral comparisons and graph architecture. CheckList relates input edits to invariance and directional expectations (Ribeiro et al., 2020). Our controlled labels compare reference outcomes with the ordered outcomes of two executed candidates, distinguishing lost changes, incorrect changes, and preserved invariance. Both inference branches are supplied candidate runs; neither is a trusted reference. The separate individual-claim path requires only one run.

Graph Matching Networks jointly update paired graphs through cross-graph attention and also study independent graph embeddings (Li et al., 2019). Our corresponding program encoders share weights across runs, with source–node alignment within each run before pooled features are compared. GraphGPS combines local message passing, global attention, and positional or structural encodings (Rampášek et al., 2022), providing related architectural components. Gradients train our neural representations and prediction heads; candidate programs and symbolic outcomes remain fixed.

Interpretation uncertainty and information measures. Paraphrase-based semantic parsing uses alternative linguistic realizations to resolve ambiguous input (Arthur et al., 2015). ConformalNL2LTL constructs sets of temporal-logic translations with coverage guarantees under its calibration assumptions (Sundarsingh et al., 2025). Our semantics retains admissible source interpretations and keeps interpretation uncertainty separate from unsupported assertions and execution failures.

Information bottleneck methods formalize task-relevant compression (Tishby et al., 2000; Alemi et al., 2017). Semantic entropy measures uncertainty across answer meanings to detect confabula-

tions (Farquhar et al., 2024). Semantic Entropy Probes predict high versus low semantic entropy from hidden states of a single generation (Kossen et al., 2024). Their target differs from reference-outcome recoverability given a policy graph and question.

Information-theoretic and minimum-description-length probing relate recoverability to predictive effort (Pimentel et al., 2020; Voita & Titov, 2020). Here, description length concerns labels rather than graph size. A successful outcome probe cannot certify candidate execution or source support. The trained execution head predicts candidate behavior; an independent diagnostic probe would assess reference-outcome recoverability after freezing the representation. The latter is an analytical interface in this paper, not a reported empirical result.

Intervention object and attribution. CausalGaze constructs graphs from token hidden states and attention maps and applies gradient-guided counterfactual edge refinement (Kong et al., 2026). Our interventions instead replace recorded policy, query, or downstream artifacts under fixed context. Checkpoint scores nominate suspected divergence; validated replacements and controlled replay test contribution. These findings concern observed artifacts and the tested repairs, without recovering hidden LLM reasoning or establishing a unique cause.

Scope of the contribution. Each form of evidence supports a different conclusion. Proofs and finite constructions establish the information identities and semantic distinctions. Frozen held-out predictions support detection and calibration measurements. Separate development evaluations assess the extended detectors and checkpoint heads. Deterministic replay establishes contribution within the tested repair families. These results establish neither human-validated attribution nor superiority over matched learned architectures.

The framework connects policy edits, semantic sensitivity, and claim support using established attention mechanisms, graph transformers, information measures, and selective prediction.