Faithful Autoformalization via Roundtrip Verification and Repair

Anonymous ACL submission

Abstract

When an LLM formalizes natural language, how do we know the output is faithful? We propose a roundtrip verification approach which does not require ground-truth annotations: formalize a statement, translate the result back to natural language, re-formalize, and use a formal tool to check logical equivalence. When the two formalizations agree, this provides evidence of a faithful formalization. When they disagree, a stage-level diagnosis localizes the error to a specific translation step, and a scoped repair operator attempts to correct that step. We evaluate the framework on two statutory domains (the Texas Transportation Code and the Texas Parks and Wildlife Code) using two LLMs (Claude Opus 4.6 and GPT-5.2) with three repair baselines. Diagnosis-guided scoped repair is the most effective method, with effectiveness contingent on the reliability of the diagnosis function. Across both domains and both models, under our full repair system, rules that fail the equivalence check show 1.4×-2.5× more NLI drift than rules that pass it.

1 Introduction

Autoformalization, the translation of natural-language statements into formal, machine-checkable representations, is a rapidly growing area of NLP research (Wu et al., 2022). Yet a fundamental question remains largely unaddressed: *when an LLM produces a formalization, how do we know it faithfully captures the meaning of the original text?* Existing evaluations rely on syntactic validity or downstream task success, neither of which directly tests semantic fidelity. Furthermore, when errors are found, repair is limited to end-to-end regeneration with no diagnosis of where meaning was lost (§2).

We propose a framework that addresses both problems (verification and repair) without requiring ground-truth formalizations. The key idea is simple (Figure 1): given a natural-language specification x, we formalize it (T_1), translate the result back to natural language (T_2), and then re-formalize the result (T_3). We call each of these translation steps a *stage*. If the pipeline preserves meaning, the initial and final formalizations should be semantically equivalent. When they disagree, the point of divergence localizes the error to a specific stage, enabling both valuable diagnostic information as well as the possibility of *scoped repair*: attempting to correct only the faulty component rather than regenerating everything. The framework is agnostic to the choice of the formalism: it applies whenever a target formalism admits an equivalence check, e.g., automated reasoning tools for logical formulas, type-checkers for proof assistants, test suites for code, etc.

We evaluate the framework on two statutory domains: 150 rules from the Texas Transportation Code and 77 rules from the Texas Parks and Wildlife Code. We use two LLMs, Claude Opus 4.6 (Anthropic) and GPT-5.2 (OpenAI). The target formalism is based on many-sorted first-order logic, with equivalence checked by an SMT solver (Barrett et al., 2021). We compare our diagnosis-guided scoped repair against three baselines: no repair, repair with random stage selection, and full-pipeline regeneration. The results yield two principal findings:

1. **Formal equivalence predicts semantic faithfulness.** Across both domains and both models, under our full repair system, rules that fail the equivalence check show 1.4×-2.5× more NLI drift than rules that pass it.

2. **Diagnosis-guided scoped repair achieves the highest verified-equivalence rate at the lowest cost when the diagnosis step is reliable.** With Claude performing the pipeline, repair, and diagnosis, scoped repair leads on both domains. With GPT in those roles, diagnosis is unreliable, causing scoped repair

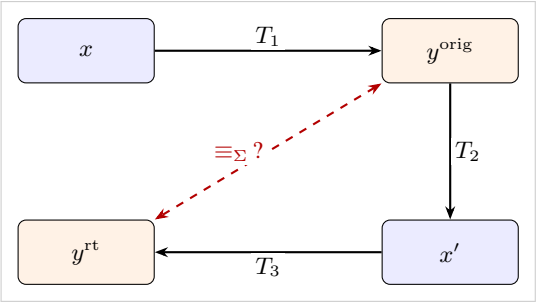

Figure 1: Roundtrip loop. An input x is formalized (T_1), reconstructed back to natural language (T_2), and re-formalized (T_3). The diagonal compares y^{orig} and y^{rt} for semantic equivalence under Σ.

to lose its lead. Replacing only the diagnosis step with Claude (keeping GPT for the pipeline and repair) restores scoped repair to a clear lead at lower cost on both domains.

Our contributions are: (i) A roundtrip verification framework that signals faithfulness of autoformalization without requiring ground-truth formalizations, instantiated with SMT in our experiments and applicable in principle to any target formalism with an equivalence check. (ii) A stage-level diagnosis and scoped repair procedure that localizes errors to individual translation steps. Repair is driven by formal equivalence alone. A bidirectional NLI signal serves as an independent post-hoc semantic check. (iii) An empirical study across two statutory domains and two model families that identifies the diagnosis step (not the pipeline or repair operators) as the bottleneck for scoped repair.

2 Related Work

Autoformalization (Wu et al., 2022) has produced NL-formal benchmarks for proof assistants (Zheng et al., 2022; Azerbayev et al., 2023; Ying et al., 2024; Gao et al., 2024; Jiang et al., 2023a) and methods that use prover or compiler feedback to improve formalization quality (Jiang et al., 2023b; Lu et al., 2024; Yang et al., 2023; Murphy et al., 2024). Concurrent work formalizes non-mathematical text (Manas et al., 2024).

Verification-integrated repair. Several systems couple verification with repair: Clover checks mutual consistency among LLM-generated code, docstrings, and formal annotations via a Dafny verifier (Sun et al., 2023), but it assumes the existence of the formalization from the start. ProofBridge iteratively repairs Lean proofs from type-checker feedback (Jana et al., 2025). DeepSeek-Prover-V1.5 uses RL with proof-assistant rewards (Xin et al., 2024). Self-debugging repairs code from execution results (Chen et al., 2023). These systems treat verifier feedback as a binary accept/reject signal, but they do not diagnose *which stage* of a multi-step pipeline introduced the error. A parallel self-refinement line (Madaan et al., 2023; Shinn et al., 2023; Pan et al., 2024) forgoes external verification entirely and uses the LLM as its own feedback source.

Faithfulness without ground truth. Faithfulness in generated text is widely studied in NLG (Maynez et al., 2020; Manakul et al., 2023; Min et al., 2023) but typically over natural-language text without a formal target. Round-trip correctness has been used as an evaluation signal in code generation when checking equivalence via test suites (Allamanis et al., 2024). In autoformalization, recent work uses re-informalization combined with consistency checks: Li et al. (2024) rank candidates using a mix of symbolic equivalence and embedding similarity over re-informalized text, while Chen et al. (2026) train a model to self-evaluate semantic fidelity and iteratively self-correct. These approaches target mathematics, where the target formalism provides a natural oracle (a stated theorem can be type-checked against its proof). Regulatory text offers no such oracle, motivating our use of formal SMT equivalence over the round-trip artifacts together with stage-level diagnosis and scoped repair, rather than candidate selection or single-pass self-correction.

3 Roundtrip Autoformalization Framework

Target formalism and notation. Let $\mathcal{X}$ denote the set of natural-language strings. We use Σ to represent the target formalism. The framework is agnostic to the specific choice of Σ: all that is required is for there to be a notion of well-formed expressions and a notion of formal equivalence. Let $\mathcal{Y}_\Sigma$ denote the set of well-formed $\Sigma-$expressions. We write $y_a \equiv_\Sigma y_b$ when two formulas $y_a, y_b \in \mathcal{Y}_\Sigma$ are equivalent according to Σ. For example, Σ could be first-order logic with the standard notion of logical equivalence, or it could be a formalism based on dependent type theory such as that used by the Lean 4 theorem prover (de Moura and Ullrich, 2021).

Three translation stages. We model roundtrip autoformalization as a composition of three translation functions:

$$T_1 : \mathcal{X} \to \mathcal{Y}_\Sigma \quad (1)$$
$$T_2 : \mathcal{Y}_\Sigma \to \mathcal{X} \quad (2)$$
$$T_3 : \mathcal{X} \to \mathcal{Y}_\Sigma \quad (3)$$

where T_1 performs *autoformalization*, T_2 performs *back-translation*, and T_3 performs *re-formalization*. All three stages operate under the same fixed target formalism Σ. Each T_i is assumed to be *stateless*: successive calls are mutually independent, with no shared memory or parameter updates between stages. The framework is agnostic to how each T_i is realized (e.g., an LLM, a rule-based translator, or a human annotator). Note that T_1 and T_3 share the same type: both translate from natural language strings to well-formed Σ expressions. We consider them distinct to maintain generality and also so that stage-level diagnosis and scoped repair can refer unambiguously to the pipeline stage at which an error originates.

Roundtrip instance. Consider again Figure 1. Given an input $x \in \mathcal{X}$, the roundtrip pipeline produces:

$$y^{\text{orig}} = T_1(x) \in \mathcal{Y}_\Sigma \quad (4)$$
$$x' = T_2(y^{\text{orig}}) \in \mathcal{X} \quad (5)$$
$$y^{\text{rt}} = T_3(x') \in \mathcal{Y}_\Sigma. \quad (6)$$

We refer to y^{orig} as the *original formalization* and to y^{rt} as the *roundtrip formalization*.

Why roundtrip? The roundtrip construction yields a verification signal *without requiring ground-truth formalizations*. If T_1 faithfully captures the meaning of x, then back-translating and re-formalizing should recover the same formal semantics. When the two formalizations disagree, the point of divergence localizes the error.

Formal equivalence. The primary verification question is whether $y^{\text{orig}} \equiv_\Sigma y^{\text{rt}}$. If not, at least one translation stage has introduced or lost meaning, triggering a diagnosis step in which an LLM judge examines all four pipeline artifacts $(x, y^{\text{orig}}, x', y^{\text{rt}})$ to identify the responsible stage and produce a scoped explanation for repair (§4). Note that if the roundtrip is consistent with the original (i.e., $y^{\text{orig}} \equiv_\Sigma y^{\text{rt}}$), correctness is still not guaranteed: the pipeline may stabilize at a semantically different fixed point where both formalizations agree yet neither faithfully represents x. We therefore treat formal consistency as *necessary but not sufficient*. In §5, we discuss how to supplement roundtrip verification with an additional NLI-based check.

Syntactic well-formedness as prerequisite. Before semantic equivalence can be checked, each formula must be syntactically valid according to Σ. LLM-generated formulas occasionally violate grammar or type constraints. We allow up to five LLM-based correction attempts per formula, which in our experience suffices to ensure that at least one parsable encoding is produced.

4 Stage-Based Diagnosis and Iterative Repair

When $y^{\text{orig}} \not\equiv_\Sigma y^{\text{rt}}$, the roundtrip instance is formally inconsistent, implying that at least one of the translation stages introduced semantic drift. However, the equivalence test alone does not identify which stage is responsible. We therefore introduce a stage-based diagnosis and repair procedure that localizes the failure and applies a targeted repair operator (Figure 2).

4.1 First-Failure Diagnosis

First-failed stage. We view the pipeline as an ordered sequence of stages $T_1 \prec T_2 \prec T_3$. Intuitively, an error introduced earlier may propagate downstream, so we aim to identify the earliest stage at which the pipeline ceases to preserve semantics. Formally specifying natural-language semantics is beyond the scope of this work. Instead, diagnosis operates over the observable artifacts $(x, y^{\text{orig}}, x', y^{\text{rt}})$ and returns an index in $\{1, 2, 3\}$.

Diagnosis function. Let

$$D : \mathcal{X} \times \mathcal{Y}_\Sigma \times \mathcal{X} \times \mathcal{Y}_\Sigma \to \{1, 2, 3\} \times \mathcal{E} \quad (7)$$

be a diagnosis procedure which, given the roundtrip artifacts, predicts the first failed stage and produces an *explanation* $e \in \mathcal{E}$, where $\mathcal{E}$ is the set of possible explanations. In our system, D is implemented as a constrained LLM-based judge and $\mathcal{E}$ is simply $\mathcal{X}$, the set of natural language strings. We enforce a *sequential checking protocol*: D must examine stages strictly in order (first comparing x with y^{orig} for Stage 1, then y^{orig} with x' for Stage 2, and finally x' with y^{rt} for Stage 3), halting at the first detected inconsistency.

Scoped diagnostic reasoning. Critically, the explanation e returned by D is constrained to reference *only* the artifacts relevant to the diagnosed stage. For instance, if Stage 2 is diagnosed as faulty, the explanation must describe the mismatch between y^{orig} and x' without mentioning x or y^{rt}. This scoping ensures that the diagnostic feedback can be directly incorporated into the corresponding repair prompt without introducing confounding information from other pipeline stages.

4.2 Targeted Repair Operators

Repair operators. A key design principle is that repairing stage i invalidates all downstream artifacts produced by later stages; after repair, we therefore regenerate all subsequent stages to maintain a coherent pipeline state. Each repair operator receives the relevant artifacts together with the diagnostic explanation e produced by D:

$$R_1 : \mathcal{X} \times \mathcal{Y}_\Sigma \times \mathcal{E} \to \mathcal{Y}_\Sigma \tag{8}$$

$$R_2 : \mathcal{Y}_\Sigma \times \mathcal{X} \times \mathcal{E} \to \mathcal{X} \tag{9}$$

$$R_3 : \mathcal{X} \times \mathcal{Y}_\Sigma \times \mathcal{E} \to \mathcal{Y}_\Sigma \tag{10}$$

where R_1 repairs the original formalization, R_2 repairs the reconstructed NL, and R_3 repairs the roundtrip formalization. Each R_i is instantiated as an LLM call operating under the same target formalism Σ. The diagnostic explanation e is injected into the repair prompt, instructing the model to address the *specific* issues identified during diagnosis rather than regenerating blindly. This feedback-driven design enables targeted corrections: the repair prompt explicitly states what semantic mismatch was detected, and the LLM is required to resolve that mismatch while preserving other aspects of the encoding.

4.3 Iterative Repair Loop

Iterative procedure. We combine verification, diagnosis, and repair into an iterative loop that attempts to reach formal self-consistency within a bounded budget. Starting from $(x, y^{\text{orig}}, x', y^{\text{rt}})$, we repeatedly: (i) check whether $y^{\text{orig}} \equiv_\Sigma y^{\text{rt}}$, (ii) if not, diagnose the first-failed stage, apply the corresponding repair operator, and regenerate downstream artifacts. The loop terminates when equivalence is achieved (success) or after a fixed maximum number of iterations (failure). We reiterate that when the procedure reports success, this does not guarantee correctness with respect to the original natural-language intent, but it does provide a measure of reassurance and confidence.

Algorithm 1 Roundtrip Verification and Stage-Based Repair

Require: Natural-language input $x \in \mathcal{X}$, formalism Σ, max iterations K

$y^{\text{orig}} \leftarrow T_1(x)$
$x' \leftarrow T_2(y^{\text{orig}})$
$y^{\text{rt}} \leftarrow T_3(x')$
for $k = 1$ to K **do**
  if $y^{\text{orig}} \equiv_\Sigma y^{\text{rt}}$ **then**
    Return SUCCESS, $(y^{\text{orig}}, x', y^{\text{rt}})$
    return
  end if
  $i \leftarrow D(x, y^{\text{orig}}, x', y^{\text{rt}})$ {first-failed stage}
  if $i = 1$ **then**
    $y^{\text{orig}} \leftarrow R_1(x, y^{\text{orig}})$
    $x' \leftarrow T_2(y^{\text{orig}})$
    $y^{\text{rt}} \leftarrow T_3(x')$
  else if $i = 2$ **then**
    $x' \leftarrow R_2(y^{\text{orig}}, x')$
    $y^{\text{rt}} \leftarrow T_3(x')$
  else if $i = 3$ **then**
    $y^{\text{rt}} \leftarrow R_3(x', y^{\text{rt}})$
  end if
end for
Return FAILURE, $(y^{\text{orig}}, x', y^{\text{rt}})$

5 Auxiliary Semantic Signal

Bidirectional NLI as a semantic proxy. We use natural language inference (NLI) to compare the original rule x with the reconstructed natural-language description $x' = T_2(y^{\text{orig}})$. NLI has been used as a faithfulness proxy across NLG tasks (Falke et al., 2019; Honovich et al., 2022; Laban et al., 2022). If the roundtrip preserves semantics, x and x' should be mutually entailing: x entails x' (nothing lost) and x' entails x (nothing spuriously added). We operationalize this using a BART-large model fine-tuned on MultiNLI, computing entailment probabilities in both directions:

$$E_{\rightarrow} = P(\text{entailment} \mid x, x') \tag{11}$$

$$E_{\leftarrow} = P(\text{entailment} \mid x', x) \tag{12}$$

along with contradiction probabilities $C_{\rightarrow}$ and $C_{\leftarrow}$.

Diagnostic categories. We derive $E_{\min} = \min(E_{\rightarrow}, E_{\leftarrow})$ for mutual entailment and $C_{\max} = \max(C_{\rightarrow}, C_{\leftarrow})$ for contradiction. Each rule pair is classified into one of six categories via the decision tree in Figure 3 (first match wins). The six categories (*Contradiction*, *Equivalent*, *Strengthened*,

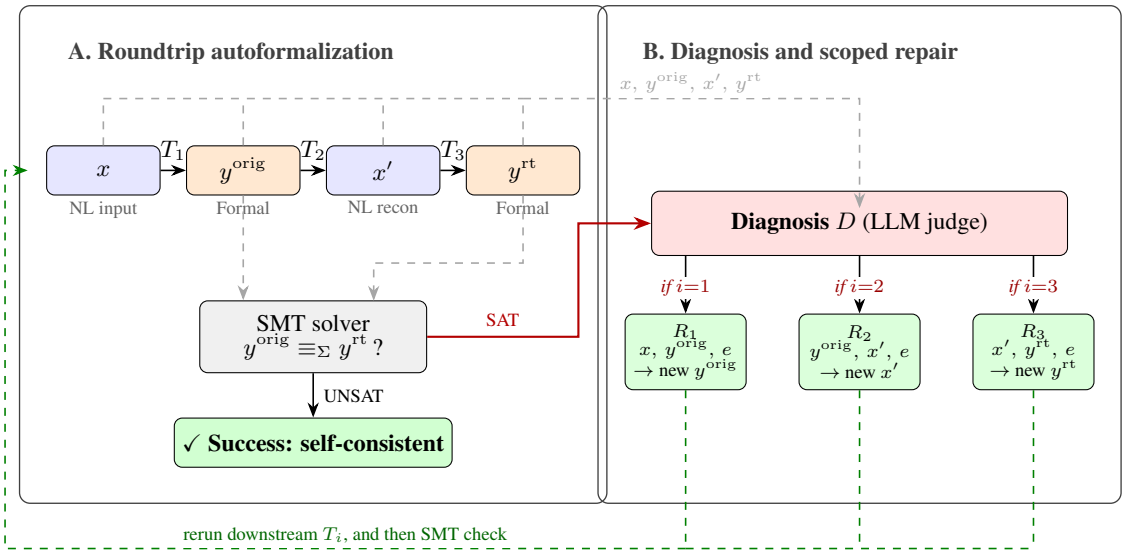

Figure 2: The roundtrip autoformalization framework. **(A)** The pipeline produces two formal encodings of x and an SMT solver checks them for equivalence. **(B)** On disagreement, diagnosis D localizes the failed stage and repair operator R_i corrects it before re-checking.

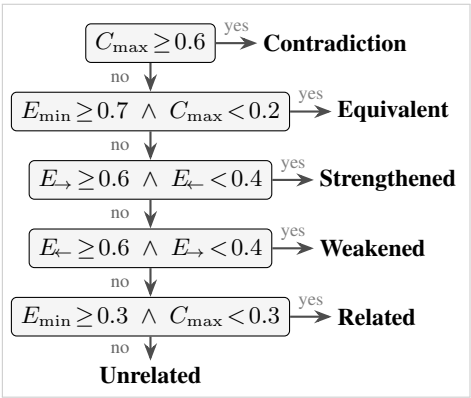

Figure 3: NLI diagnostic decision tree (first match wins). Thresholds are conservative. *Equivalent* demands strong bidirectional entailment (≥ 0.7) with near-zero contradiction (≤ 0.2), while *Contradiction* requires high conflict (≥ 0.6).

Weakened, *Related*, *Unrelated*) capture not only whether meaning is preserved but *how* it drifts, a distinction that a binary test would obscure.

Diagnostic, not supervisory. NLI scores are used *only for post-hoc analysis*, not as optimization targets. The repair loop (§4) operates solely on formal equivalence. NLI serves as an independent diagnostic lens, allowing us to ask: *when formal self-consistency is achieved, does semantic fidelity follow?* This separation avoids optimizing for a noisy proxy while still providing a meaningful signal. In §7, we cross-tabulate formal equivalence with NLI categories to reveal where formal consistency diverges from semantic alignment.

6 Experimental Setup

6.1 Domains and Datasets

From law to logic. We instantiate the framework on statutory regulation, which combines natural-language ambiguity with precise operational requirements. We evaluate on two corpora: traffic law and wildlife regulation. Any statutory domain admitting a formal equivalence check could serve equally well (§3).

Traffic corpus. We evaluate on 150 rules drawn from the Texas Transportation Code. These rules govern vehicle behavior on roadways, covering lane positioning, passing maneuvers, signaling requirements, speed limits, and interactions with special vehicles (e.g., streetcars, school buses, emergency vehicles).

Wildlife corpus. We evaluate on 77 rules drawn from the Texas Parks and Wildlife Code, a second statutory domain. These rules cover hunting and fishing licensing, season and bag-limit restrictions, equipment and method restrictions, and protected-species rules.

6.2 Models and Configuration

Formalism. A *Satisfiability Modulo Theories* (SMT) solver is a tool that can formally determine whether a set of first-order formulas is *satisfiable* (i.e., whether there exists an interpretation that makes all of the formulas true) with respect to some background theory T (Barrett et al., 2010).

An SMT solver returns one of two verdicts: SAT (a satisfying interpretation exists) or UNSAT (no such interpretation exists). SMT solvers can check whether two formulas φ and ψ are logically equivalent (with respect to a theory T), by checking the satisfiability of $\neg(\varphi = \psi)$. If the result is UNSAT, no interpretation can distinguish the two formulas, certifying equivalence. If SAT, the solver can return a counterexample demonstrating the difference. We construct a *domain schema* for each corpus, a set of SMT declarations and definitions that provide a fixed vocabulary for that domain's concepts. Schemas are produced semi-automatically: we draft a sketch by hand, scale it up with an LLM, and curate the result against a list of design principles to avoid common smells (Appendix B). We use the Z3 SMT solver (de Moura and Bjørner, 2008) (version 4.15.3) to check equivalence, and all queries resolve in under one second.

Language models and repair. To assess framework generalizability across model families, we run the full pipeline with two frontier LLMs: **Claude Opus 4.6** (Anthropic) and **GPT-5.2** (OpenAI). Both use temperature 0.3. Each translation stage (T_1, T_2, T_3) and repair operator (R_1, R_2, R_3) is realized as a single stateless LLM call. The domain schema is injected into every prompt, and no context is carried between calls. All stages and operators use the same prompts for both models. The iterative repair loop (§4) runs for at most $K{=}3$ iterations per rule. Each result is from a single run. The directional patterns reported in §7 hold consistently across all four domain-model cells.

NLI model. For semantic comparison between original and reconstructed natural language, we use BART-large fine-tuned on MultiNLI (Lewis et al., 2020; Williams et al., 2018) (`facebook/bart-large-mnli`). Inference is performed bidirectionally as described in §5.

Compute. Claude Opus 4.6 and GPT-5.2 are commercial APIs whose parameter counts are not publicly disclosed. NLI inference uses BART-large (approximately 0.4B parameters) on a single laptop. Per-repair LLM call counts, the dominant cost, are reported in Table 1.

6.3 Ablation Conditions

We compare four approaches:

1. **No Repair** (Baseline 0): The roundtrip pipeline runs without any repair loop. The initial equivalence check result is final.
2. **Random Stage** (Baseline 1): When the equivalence check fails, select a stage $i \in \{1, 2, 3\}$ uniformly at random, repair it, and regenerate downstream stages. Repeat up to $K{=}3$ times, stopping as soon as equivalence holds. No diagnosis is performed.
3. **Regenerate** (Baseline 2): When the equivalence check fails, regenerate the entire pipeline from the original input x and check again. Repeat up to $K{=}3$ times, stopping as soon as equivalence holds. No diagnosis is performed.
4. **Full** (Ours): We use the complete pipeline with diagnosis (§4.1) followed by targeted repair of the diagnosed stage. This is the system described in Algorithm 1.

All four approaches share the same initial roundtrip pass (identical $y^{\text{orig}}, x', y^{\text{rt}}$). They differ only in the repair strategy applied to rules where the initial equivalence check returns SAT.

The two new baselines isolate two design choices in Full. Random Stage controls for the value of *diagnosis* (which stage is selected for repair). Regenerate controls for the value of *scoping* (re-running only the diagnosed stage rather than the whole pipeline).

7 Results and Analysis

7.1 Roundtrip Equivalence and Repair

Table 1 reports final equivalence outcomes across the four domain-model cells. Without repair, equivalence ranges from 44.7 % (Traffic / Claude) to 66.2 % (Wildlife / GPT). Each repair method substantially raises equivalence in every cell.

Method comparison. The most effective repair method depends on the model, and the same pattern holds across both domains. For Claude, Full leads on both UNSAT count and cost-per-repair. For GPT with same-model diagnosis, Regenerate surpasses Full on UNSAT count, and Full is the most expensive of the three methods. This split reflects the diagnosis distribution: GPT repeatedly diagnoses T_1, which forces regenerating T_2 and T_3 downstream on every iteration (§7.2). The third row of each GPT block reports the cross-model intervention, where Claude performs only the diagnosis step. We analyze this fix in §7.5.

Domain	Model	Diagnoser	Final UNSAT count (%)				LLM calls / repair		
			None	Random	Regen	Full	Random	Regen	Full
Traffic (N=150)	Claude	Claude	67 (44.7)	100 (66.7)	97 (64.7)	**128 (85.3)**	12.27	19.90	**7.21**
	GPT	GPT	92 (61.3)	118 (78.7)	**127 (84.7)**	124 (82.7)	**9.77**	10.54	14.56
	GPT	Claude	—	—	—	**136 (90.7)**	—	—	**7.86**
Wildlife (N=77)†	Claude	Claude	48 (62.3)	60 (77.9)	63 (81.8)	**66 (85.7)**	11.67	12.20	**9.50**
	GPT	GPT	51 (66.2)	59 (76.6)	**65 (84.4)**	60 (77.9)	12.75	**11.57**	26.44
	GPT	Claude	—	—	—	**72 (94.7)**	—	—	**6.05**

Table 1: Roundtrip equivalence outcomes across two domains and four repair conditions. The **Model** column performs pipeline and repair. The **Diagnoser** performs diagnosis. Bold marks the per-row best. †One Wildlife rule was skipped at baseline and is excluded from post-repair counts.

Domain / Model	Method	UNSAT	SAT	Ratio
Traffic / Claude	Random	14.0%	20.0%	1.43
	Regenerate	12.4%	21.2%	1.71
	Full	9.4%	13.6%	1.45
Traffic / GPT	Random	15.3%	28.1%	1.84
	Regenerate	15.0%	26.1%	1.74
	Full	13.7%	34.6%	2.53
	Full (Claude dx)*	12.5%	35.7%	2.86
Wildlife / Claude	Random	25.0%	35.3%	1.41
	Regenerate	25.4%	28.6%	1.13
	Full	24.2%	45.5%	1.88
Wildlife / GPT	Random	18.6%	28.6%	1.54
	Regenerate	20.0%	55.6%	2.78
	Full	20.0%	37.5%	1.88
	Full (Claude dx)*	19.4%	66.7%‡	3.43‡
Pooled		**16.2%**	**28.1%**	**1.74**

Table 2: Post-repair drift rate (share of rules NLI-classified as Unrelated or Contradiction), split by SMT verdict. Ratio is SAT drift divided by UNSAT drift. *Cross-model: GPT pipeline + Claude diagnosis (§7.5). ‡Wildlife / Claude-dx SAT pool has only 3 rules; the rate is noisy.

7.2 Stage Diagnosis Distribution

The two models diagnose pipeline failures very differently, and the pattern is consistent across both domains (Table 3, Appendix C). Claude distributes its diagnoses across all three stages. GPT collapses almost all of its diagnoses onto T_1. The Random baseline distributes approximately uniformly across stages in every cell, so the concentration is a property of the diagnosis function, not the data.

Repairing T_1 cascades: it forces a fresh T_2 and a fresh T_3 on every iteration, while repairing T_3 does not. GPT's T_1-heavy diagnosis therefore behaves like full-pipeline regeneration plus a diagnosis call, which explains Full's higher cost than Regenerate for GPT (Table 1). §7.5 tests whether this T_1 concentration is a property of the protocol or of the model.

7.3 Formal vs. Semantic Alignment

We examined the rate of every NLI category in each pool (Appendix D, Table 4). Related and Weakened distribute similarly between UNSAT and SAT. Strengthened concentrates in UNSAT. This reflects an artifact of the back-translation: the reconstructed NL tends to use the schema's predicate names verbatim, which produces wordier text than the original lawyer-written rule. NLI interprets the wordier reconstruction as a stricter version of the original, so NLI-Strengthened arises even when the formalization is faithful. Only Unrelated and Contradiction discriminate cleanly between UNSAT and SAT, motivating their use as our Drift measure.

Formal equivalence predicts semantic faithfulness. Figure 4 shows that under Full repair, SAT rules drift 1.45×-2.52× more often than UNSAT rules across the four domain-model cells. Pooled across these four Full configurations, SAT rules drift 2.03 times more often than UNSAT rules. The same pattern holds when baselines are included: pooled across all 14 method-cell combinations in Table 2, SAT rules still drift 1.74 times more often than UNSAT rules. The pattern is consistent across both model families and both statutory domains. Formal equivalence does not guarantee semantic fidelity, but it predicts it.

7.4 Residual Failure Analysis

A rule is *residual* if none of Random, Regenerate, or Full repairs it within K=3 iterations. 26 unique rules are residual across the four cells. Manual inspection identifies four recurring patterns of intrinsic rule difficulty: **schema vocabulary gaps** (the schema lacks a needed predicate, so T_1 and T_3 approximate with different substitutes), **compound conditions** (disjunctive or nested premises where T_1 and T_3 encode the AND/OR structure

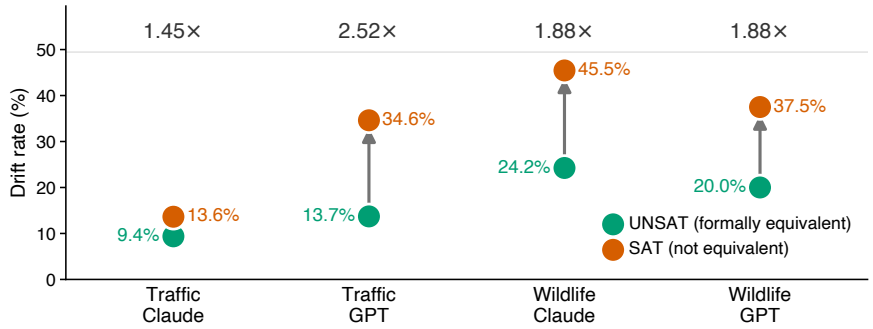

Figure 4: NLI drift rate among UNSAT (formally equivalent) and SAT (not equivalent) post-repair rules under Full repair, one bar per domain-model cell. The annotation above each pair is the SAT-to-UNSAT drift ratio.

differently), and **exception clauses** (rules that suspend an obligation under a condition, where T_2 back-translates the exception as a positive restatement and T_3 then encodes the inverted meaning), and **conditional thresholds** (numeric thresholds that depend on another variable, where the pipeline often drops the dependency). The bottleneck is intrinsic rule structure, not iteration budget.

7.5 Why does GPT's diagnosis collapse?

The GPT/Full underperformance we report in §7.2 stems from a single design point: the diagnosis function. GPT selects the first translation step T_1 in 83 % of iterations on Traffic and 97 % on Wildlife, regardless of where the actual error occurs (Table 3). Two hypotheses are compatible with this: the prompt's sequential checking protocol biases GPT toward the first stage examined, or GPT has an intrinsic preference for blaming initial formalization. We test both directly.

Non-sequential prompt. We rewrite the diagnosis prompt to drop the "examine T_1 first, then T_2, then T_3, halt at first failure" instruction and ask the judge to consider all three stages together. Rerunning diagnosis on the existing GPT SAT pools, T_1 selection drops from 85 % to 9 % on Traffic and from 96 % to 16 % on Wildlife. The protocol explains most of the collapse.

Cross-model diagnosis. We rerun the Full repair loop on the existing GPT pipeline outputs but use Claude as the diagnosis judge while GPT still performs the actual repair. Final UNSAT rises from 124 to 136 on Traffic and from 60 to 72 on Wildlife. Cost-per-repair drops from 14.56 to 7.86 and from 26.44 to 6.05 (Table 1, last two rows). On both domains, GPT-pipeline + Claude-diagnose + GPT-repair exceeds Claude/Full on UNSAT count and matches or beats it on cost. The cross-model intervention preserves the faithfulness signal: UNSAT drift is 12.5 % (Traffic) and 19.4 % (Wildlife), comparable to the same-model rates of 13.7 % and 20.0 % (Table 2, last row of each GPT block). The diagnosis function was the bottleneck, not the pipeline or the repair operators.

8 Conclusion

We evaluated a roundtrip verification and scoped repair framework on two statutory domains and two model families. Two findings stand out. First, formal equivalence is a useful predictor of semantic faithfulness: across the four Full configurations spanning both domains and both models, SAT rules show 1.4×-2.5× more NLI drift than UNSAT rules. Second, diagnosis-guided scoped repair achieves the highest verified-equivalence rate at the lowest cost when the diagnosis function is reliable. On Claude pipelines, same-model scoped repair wins on both domains. On GPT pipelines, the diagnosis step is the bottleneck for scoped repair (not the pipeline or repair operators): replacing only the diagnoser with a different model restores scoped repair's lead at lower cost (§7.5). About 16 % of formally equivalent rules still drift on NLI. Future work could integrate additional checks into the repair loop, such as NLI or an LLM-based check, rather than relying on formal equivalence alone.

Limitations

Domain scope. Both corpora are English statutory text from Texas. Mathematical autoformalization benchmarks typically operate one theorem at a time, where a human author can read the formal statement and verify it against the informal one. Statutory text involves a large domain schema (close to 200 predicates) and operational ambigu-

ity, so hand verification at scale is impractical and a ground-truth-free faithfulness signal is most directly motivated by this setting. We do not claim the framework transfers to other genres such as code or scientific text without further evaluation.

Verification is heuristic, not a proof. The roundtrip check detects when $T_1(x)$ and $T_3(T_2(T_1(x)))$ disagree, but it cannot detect errors that affect both stages in the same way. For example, if T_1 silently drops a conjunct of x and T_3 reproduces the same omission from the back-translation, the equivalence check passes despite both being wrong. The NLI cross-check (§5) is a partial mitigation, but it relies on a general-purpose model with known lexical-overlap biases (McCoy et al., 2019) on legal text, so its drift signal is itself noisy.

Rules are formalized in isolation. Statutory text can be heavily cross-referenced. A rule may incorporate definitions from another section, suspend obligations stated elsewhere, or impose conditions on rules outside its own clause. Our pipeline formalizes one rule at a time and treats each rule as self-contained. Faithful formalization of densely cross-referenced rules likely requires presenting the connected cluster of rules together as input, rather than a single rule node. We leave this to future work.

Risks

Autoformalization outputs generated by this system, including those that pass roundtrip verification, should not be used in safety-critical applications, including but not limited to autonomous systems, medical devices, or safety case documentation, without independent review by a qualified formal methods expert. Regarding environmental impact, the pipeline relies on API-based LLM inference and a locally-run SMT solver. We did not train or fine-tune any models, so the compute footprint is bounded by inference calls.

Use of AI Assistants

Claude Code was carefully used to assist with code implementation, LaTeX formatting, and paraphrasing during paper writing. All scientific content, experimental design, analysis, and claims are entirely the authors' own.

References

Miltiadis Allamanis, Sheena Panthaplackel, and Pengcheng Yin. 2024. Unsupervised evaluation of code LLMs with round-trip correctness. *Preprint*, arXiv:2402.08699.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. 2023. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. ArXiv:2302.12433.

Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. 2021. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability, Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 33, pages 825–885. IOS Press.

Clark Barrett, Aaron Stump, and Cesare Tinelli. 2010. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*.

Guoxin Chen, Jing Wu, Xinjie Chen, Wayne Xin Zhao, Ruihua Song, Chengxi Li, Kai Fan, Dayiheng Liu, and Minpeng Liao. 2026. ReForm: Reflective autoformalization with prospective bounded sequence optimization. In *International Conference on Learning Representations (ICLR)*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 337–340.

Leonardo de Moura and Sebastian Ullrich. 2021. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, pages 625–635. Springer.

Tobias Falke, Leonardo F. R. Ribeiro, Prasetya Ajie Utama, Ido Dagan, and Iryna Gurevych. 2019. Ranking generated summaries by correctness: An interesting but challenging application for natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2214–2220, Florence, Italy. Association for Computational Linguistics.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. 2024. Herald: A natural language annotated lean 4 dataset. *arXiv preprint arXiv:2410.10878*.

Or Honovich, Roee Aharoni, Jonathan Herzig, Hagai Taitelbaum, Doron Kukliansy, Vered Cohen, Thomas Scialom, Idan Szpektor, Avinatan Hassidim, and Yossi Matias. 2022. TRUE: Re-evaluating factual

consistency evaluation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3905–3920, Seattle, United States. Association for Computational Linguistics.

Prithwish Jana, Kaan Kale, Ahmet Ege Tanriverdi, Cruise Song, Sriram Vishwanath, and Vijay Ganesh. 2025. ProofBridge: Auto-formalization of natural language proofs in Lean via joint embeddings. *arXiv preprint arXiv:2510.15681*.

Albert Q. Jiang, Wenda Li, and Mateja Jamnik. 2023a. Multilingual mathematical autoformalization. ArXiv:2311.03755.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. 2023b. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations (ICLR)*. ArXiv:2210.12283.

Philippe Laban, Tobias Schnabel, Paul N. Bennett, and Marti A. Hearst. 2022. SummaC: Re-visiting NLI-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics*, 10:163–177.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Xian Zhang, Fan Yang, and Xiaoxing Ma. 2024. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jianqiao Lu, Zhengying Liu, Yingjia Wan, Yinya Huang, Haiming Wang, Zhicheng Yang, Jing Tang, and Zhijiang Guo. 2024. Process-driven autoformalization in Lean 4. ArXiv:2406.01940.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *Preprint*, arXiv:2303.08896.

Kumar Manas, Stefan Zwicklbauer, and Adrian Paschke. 2024. TR2MTL: LLM based framework for metric temporal logic formalization of traffic rules. ArXiv:2406.05709.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. *Preprint*, arXiv:2005.00661.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3428–3448.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. 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*, pages 12076–12100, Singapore. Association for Computational Linguistics.

Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. Autoformalizing Euclidean geometry. In *International Conference on Machine Learning (ICML)*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Chuyue Sun, Ying Sheng, Oded Padon, and Clark Barrett. 2023. Clover: Closed-loop verifiable code generation. *arXiv preprint arXiv:2310.17807*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus Norman Rabe, Charles E Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*.

Huajian Xin, Z.Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, and 1 others. 2024. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. *arXiv preprint arXiv:2408.08152*.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. 2023. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*. ArXiv:2306.15626.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. Lean workbook: A large-scale lean problem set formalized from natural language math problems. ArXiv:2406.03847.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. MiniF2F: a cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*. ArXiv:2109.00110.

A Prompt Templates

The pipeline uses prompts for each pipeline component: the three translation stages T_1, T_2, T_3, the diagnosis function D, and the three stage-specific repair operators R_1, R_2, R_3. All prompts share a common structure: a system role description, the domain schema injected verbatim, the relevant pipeline artifacts, and explicit output-format constraints. Below is the diagnosis prompt in abridged form.

Listing 1: Diagnosis prompt (abridged).

```
You are analyzing a 3-arrow translation
pipeline that converts natural language
into SMT-LIB encodings:
  ARROW 1: Original NL -> Phase 1 SMT
  ARROW 2: Phase 1 SMT -> Reconstructed NL
  ARROW 3: Reconstructed NL -> Phase 3 SMT

Z3 returned SAT, so Phase 1 and Phase 3
encodings differ. Identify the FIRST arrow
that introduced the divergence.

Procedure:
1. Check Arrow 1. If Phase 1 SMT does not
   match Original NL semantics, stop and
   report Arrow 1.
2. Else check Arrow 2. If Reconstructed NL
   does not describe Phase 1 SMT, stop and
   report Arrow 2.
3. Else report Arrow 3.

ARTIFACTS:
  Original NL:        {original_nl}
  Phase 1 SMT:        {phase1_smt}
  Reconstructed NL:   {reconstructed_nl}
  Phase 3 SMT:        {phase3_smt}

Respond in this format:
  FIRST_FAILED_ARROW: [1, 2, or 3]
  REASONING: [a short paragraph; reference
    only the artifacts adjacent to the
    diagnosed arrow]
```

The repair prompts inject the diagnostic reasoning as a `DIAGNOSTIC FEEDBACK` field and instruct the model to change only what is necessary to address the identified problem.

B Schemas

Construction process. Each domain schema is produced semi-automatically. We hand-draft an initial schema sketch on a small set of pilot rules, then ask an LLM to extend the sketch with the additional sorts, predicates, and enumerated types needed to encode the full corpus. We iterate (proposing, reviewing, refining) until coverage is satisfactory. At each round we curate the result against the design principles described below, removing or rewriting predicates that violate them.

Design principles. We curate each schema against several principles to keep predicates reusable across rules and to avoid common smells. Each predicate should encode a single concept rather than a multi-clause sentence. Action verbs and qualifiers compose separately rather than fusing into one predicate. Anchor predicates such as kind, used-device, or behavior tags are designed to fire in many rules. Discrimination among related entities uses one sort with a kind enumeration rather than multiple sorts. Numeric thresholds are exposed as Real-valued functions rather than baked into predicate names. Opaque named enumerations (for example, long lists of protected species) collapse into a single black-box predicate. Time-varying predicates take an integer time argument in the last position; time-independent predicates omit it.

Excerpts. Short illustrative excerpts of both schemas are shown below.

Listing 2: Traffic schema (excerpt).

```
; Sorts
(declare-sort Vehicle 0)
(declare-sort Roadway 0)

; Static properties
(declare-fun kind (Vehicle) VehicleKind)
(declare-fun roadway_kind (Roadway) RoadwayKind)

; Time-varying predicates (Int = time)
(declare-fun on_roadway (Vehicle Roadway Int) Bool)
(declare-fun speed (Vehicle Int) Real)
```

Listing 3: Wildlife schema (excerpt).

```
; Sorts
(declare-sort Person 0)
(declare-sort Animal 0)
(declare-sort Device 0)

; Static properties
(declare-fun is_kind (Animal AnimalKind) Bool)
(declare-fun protected_by_code (Animal) Bool)

; Time-varying predicates (Int = time)
(declare-fun in_captivity (Animal Int) Bool)
(declare-fun hunts (Person Animal Int) Bool)
```

Domain	Model / Method	T_1	T_2	T_3
Traffic	Claude / Full	25%	21%	**55%**
	Claude / Random	34%	35%	32%
	GPT / Full (sequential)	**83%**	3%	14%
	GPT / Random	32%	30%	38%
	GPT / Full (non-sequential)	9%	17%	74%
	GPT / Full (Claude judge)	48%	12%	39%
Wildlife	Claude / Full	32%	8%	**60%**
	Claude / Random	35%	27%	38%
	GPT / Full (sequential)	**97%**	3%	0%
	GPT / Random	30%	19%	51%
	GPT / Full (non-sequential)	16%	20%	64%
	GPT / Full (Claude judge)	45%	12%	43%

Table 3: Stage selection distribution under the diagnosis function (Full) and the random baseline (Random). On both domains, Claude's diagnosis distributes across stages with a T_3 concentration, and GPT's diagnosis collapses onto T_1 under the sequential checking protocol. Random selection is approximately uniform. Two interventions un-collapse the GPT distribution: rewriting the prompt to drop sequential ordering (*non-sequential*, §7.5, Experiment A), or replacing GPT with Claude as the judge (*Claude judge*, §7.5, Experiment C).

NLI category	UNSAT pool (N=1275)	SAT pool (N=303)	SAT − UNSAT (pp)
Equivalent	42.4%	38.3%	−4.1
Related	9.0%	7.9%	−1.1
Strengthened	21.7%	15.8%	−5.9
Weakened	10.7%	9.9%	−0.8
Unrelated	10.0%	15.5%	+5.5
Contradiction	6.1%	12.5%	+6.4

Table 4: Per-category NLI rates in the UNSAT and SAT pools, pooled across all 14 method-cell combinations in Table 2 (3 repair methods × 4 cells, plus 2 cross-model variants). Negative values in the rightmost column mean the category is more frequent in UNSAT than in SAT. Only Unrelated and Contradiction concentrate in SAT. Strengthened concentrates in UNSAT, reflecting verbose legal back-translations of correctly formalized rules. The other categories distribute approximately evenly.

C Stage Diagnosis Details

Table 3 provides the per-domain stage selection breakdown. The Claude T_3 concentration on Traffic intensifies across iterations: 41 % at iteration 1, 67 % at iteration 2, and 71 % at iteration 3 (Figure 5). GPT's T_1 concentration on Traffic stays high throughout (84 %, 85 %, 79 %). On Wildlife, GPT diagnoses T_1 in nearly every iteration.

D NLI Category Detail

Table 4 reports the rate of each NLI category in the UNSAT and SAT pools, pooled across all 14 method-cell combinations in Table 2. Table 5 gives the full per-cell, per-method breakdown.

E Data Availability

We evaluate on two public statutory corpora, both in English. The traffic corpus consists of 150 rules derived from the Texas Transportation Code, a public-domain statutory text published by the Texas Legislature.[1] The wildlife corpus consists of 77 rules derived from the Texas Parks and Wildlife Code, also published by the Texas Legislature.[2] As U.S. state-government-authored statutes, both source texts are not subject to copyright. The source corpora contain no personally identifiable information or offensive content, as they are statutory text describing legal subjects in the abstract.

Code, data, prompts, and domain schemas accompany this submission as an anonymized repository.[3] They will be released under the MIT license upon publication. External artifacts are used under their respective open-source licenses: Z3 (MIT), BART weights (MIT), and MultiNLI (mixed permissive licenses per source). All external artifacts (Z3, BART-large-mnli, MultiNLI, Texas statutes, and commercial LLM APIs) are used consistently with their intended purposes as solvers, inference models, training data, public reference text, and language-generation services, respectively. Our released artifacts (code, data, prompts, schemas) are intended for research use on autoformalization and roundtrip-style verification.

[1] https://statutes.capitol.texas.gov/Docs/TN/htm/TN.545.htm

[2] https://statutes.capitol.texas.gov/?link=PW

[3] https://anonymous.4open.science/r/roundtrip-autoformalization-647B/

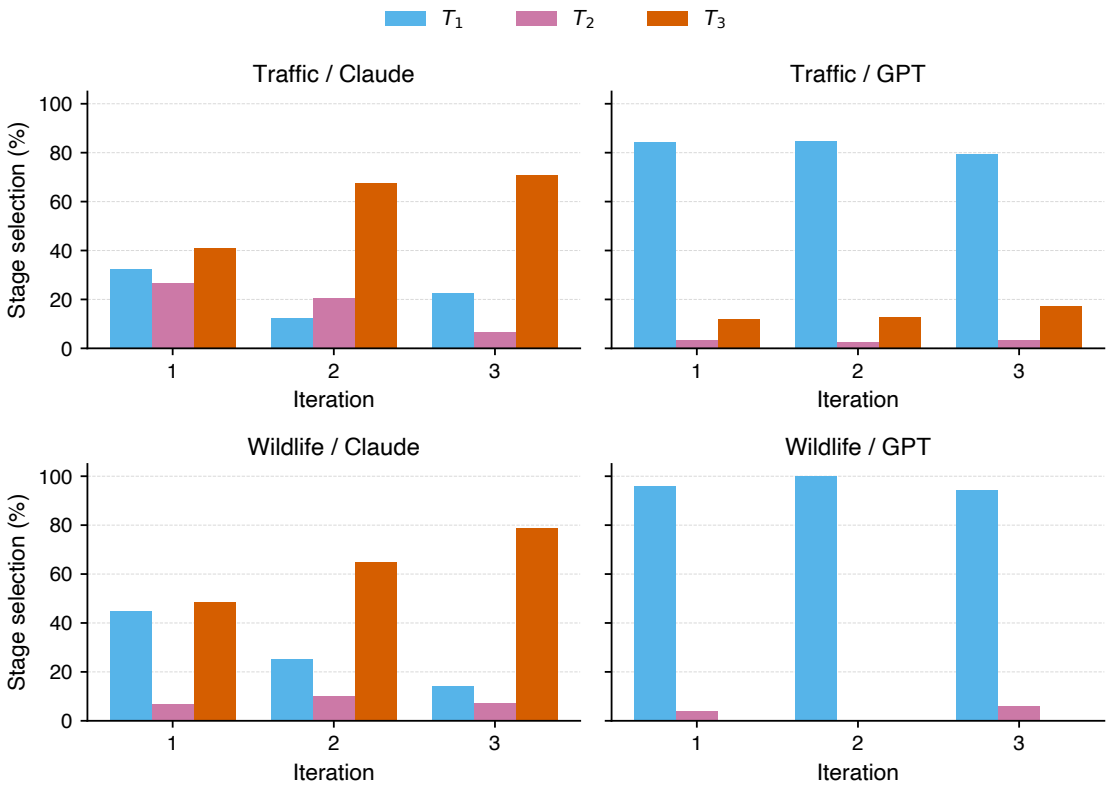

Figure 5: Per-iteration stage selection (Full condition) for each domain-model cell. Claude shifts toward T_3 in later iterations on both domains. GPT diagnoses T_1 at every iteration on both domains.

Domain / Model	Method / Verdict	Eq	Re	St	Wk	Un	Co	Total
Traffic / Claude	Random / UNSAT	55	7	18	6	10	4	100
	Random / SAT	21	5	8	6	5	5	50
	Regenerate / UNSAT	56	7	17	5	9	3	97
	Regenerate / SAT	27	1	9	4	4	7	52
	Full / UNSAT	73	13	24	6	4	8	128
	Full / SAT	13	1	4	1	1	2	22
Traffic / GPT	Random / UNSAT	62	12	12	14	9	9	118
	Random / SAT	12	3	4	4	6	3	32
	Regenerate / UNSAT	67	13	14	14	12	7	127
	Regenerate / SAT	10	1	1	5	1	5	23
	Full / UNSAT	66	12	14	15	9	8	124
	Full / SAT	10	2	4	1	5	4	26
	Full (Cdx) / UNSAT	74	16	13	16	8	9	136
	Full (Cdx) / SAT	2	1	3	3	2	3	14
Wildlife / Claude	Random / UNSAT	12	6	20	7	12	3	60
	Random / SAT	6	2	2	1	5	1	17
	Regenerate / UNSAT	13	5	24	5	12	4	63
	Regenerate / SAT	4	2	2	2	2	2	14
	Full / UNSAT	16	7	21	6	14	2	66
	Full / SAT	2	0	2	2	3	2	11
Wildlife / GPT	Random / UNSAT	10	3	24	11	6	5	59
	Random / SAT	4	2	3	1	2	2	14
	Regenerate / UNSAT	13	4	25	10	7	6	65
	Regenerate / SAT	3	1	0	0	5	0	9
	Full / UNSAT	11	3	24	10	7	5	60
	Full / SAT	2	3	5	0	4	2	16
	Full (Cdx) / UNSAT	13	7	27	11	9	5	72
	Full (Cdx) / SAT	0	0	1	0	2	0	3

Table 5: Full 6-category NLI cross-tabulations partitioned by SMT verdict, for all 14 method-cell combinations in Table 2 (3 repair methods $\times$ 4 cells, plus 2 cross-model variants on the GPT cells). Eq = Equivalent, Re = Related, St = Strengthened, Wk = Weakened, Un = Unrelated, Co = Contradiction. In UNSAT rows, shading indicates semantic quality: faithful , partial drift , unrelated , contradiction . SAT rows are uncolored.