REEXP: FROM AUTOFORMALIZATION CORRECTIONS TO TRANSFERABLE REPAIR EXPERIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoformalization enables solver-based reasoning by translating natural-language problems into executable symbolic representations, while feedback-driven refinement can correct faulty formalizations. Such refinement also produces correction histories, but a successful correction does not by itself reveal which local changes can be safely reused on new problems. We introduce REEXP, a framework that transforms failure-to-success correction histories into transferable repair experience. REEXP replays recorded corrections to isolate sufficient edits, abstracts them into typed and rebindable repair programs, and retains only repairs that demonstrate positive utility on held-out states. At inference, selected repairs are rebound and executed on compatible formalizations before standard refinement resumes. Across five logical reasoning benchmarks and four backbones, REEXP improves over within-instance solver-feedback refinement in all 20 settings, with per-backbone mean gains of 6.78–14.67 percentage points. It also outperforms retrieval-based experience reuse in most settings, with corresponding improvements in semantic faithfulness.

1 INTRODUCTION

Autoformalization uses large language models (LLMs) to translate natural-language problems into symbolic representations for solver-based reasoning (Pan et al., 2023; Olausson et al., 2023; Ye et al., 2023). Initial formalizations can be faulty, and feedback-driven refinement revises them using solver diagnostics or verification. Refinement also produces correction histories that record how faulty formalizations are transformed to recover the correct answer.

Recent work improves formalization through within-instance refinement and cross-instance reuse. Structured generation, solver feedback, and verification have been used for within-instance refinement (Putra et al., 2026; Pan et al., 2023; Kirtania et al., 2024; Amrollahi et al., 2026), while previous corrections have been reused as retrieved examples or program preferences (Hu et al., 2025; Zhou & Liu, 2026). Related work in program repair and agent systems represents reusable experience as transformations, executable skills, or structured repair records (Rolim et al., 2017; Bader et al., 2019; Wang et al., 2024; Mi et al., 2026; Mu et al., 2026). However, these approaches do not directly determine which local changes in a successful correction remain reusable across instances.

A successful correction history does not by itself identify which local changes are reusable. Different edits may recover the same source answer while having different semantic effects when transferred to another problem. As illustrated in Figure 1, both correcting an erroneous connective and deleting a condition recover the answer on P_1. After rebinding the two repairs to P_2, however, only the connective repair preserves the information required by the new problem. Even a transferable repair may provide no incremental benefit if the base refiner reaches the same result without it. Effective reuse therefore requires both isolating a transferable local repair and establishing its benefit over base refinement from comparable states.

To address these challenges, we introduce REEXP, a framework that transforms failure-to-success correction histories into reusable local repair programs. Replay isolates edit sequences sufficient to reproduce a recorded correction, while typed abstraction converts instance-specific symbols and locations into rebindable roles and anchors. Candidate repairs are admitted only when their mean paired gain over base refinement on held-out states is positive. At inference, an applicable repair is rebound and executed before base refinement continues.

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

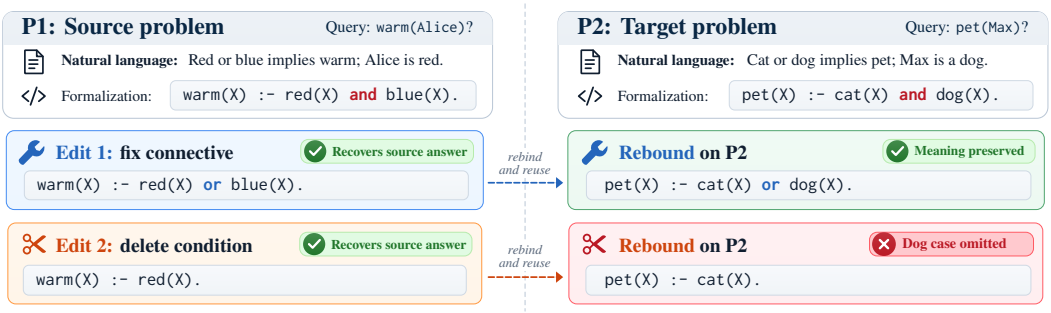


Figure 1: Illustration of ambiguous source corrections. Two corrections recover the answer on P_1, but only the connective repair preserves the relevant constraint when rebound to P_2.

To summarize, the key contributions are as follows:

1. We collect 1,558 failure-to-success correction trajectories from autoformalization refinement, identify 168 distinct repair families across five categories, and show substantial task-dependent variation in their composition.
2. We propose REEXP, which transforms correction histories into typed, rebindable repair programs through replay and abstraction, validates their incremental utility on held-out states, and selectively executes them on new formalizations.
3. Across five benchmarks and four backbones, REEXP outperforms solver-feedback refinement in all 20 settings, with mean gains of 6.78–14.67 points per backbone. It also surpasses retrieval-based reuse in most settings and improves semantic faithfulness.

2 RELATED WORK

Autoformalization for logical reasoning. Autoformalization separates language-model translation from symbolic inference (Pan et al., 2023; Olausson et al., 2023; Ye et al., 2023). Recent work improves the construction, refinement, and verification of symbolic representations. NL2Logic (Putra et al., 2026) introduces intermediate abstract syntax trees to structure formalization, while CLOVER (Ryu et al., 2025) combines compositional first-order translation with semantic verification. Logic-LM (Pan et al., 2023) refines generated representations using solver diagnostics, and Logic-LM++ (Kirtania et al., 2024) further uses pairwise assessment to evaluate proposed revisions. Roundtrip verification uses back-translation and re-formalization to check logical equivalence and guide targeted repair (Amrollahi et al., 2026), whereas Draft-and-Prune generates multiple candidate formalizations and removes contradictory or ambiguous ones before aggregating their predictions (Ni et al., 2026). These approaches primarily improve or verify the formalization of the current problem, whereas REEXP studies reuse across problems.

Reusable transformations and skills. Program repair and agent systems reuse historical edits and trajectories in structured forms. Refazer synthesizes syntactic transformations from before-and-after examples (Rolim et al., 2017). Getafix (Bader et al., 2019) organizes repair patterns hierarchically and ranks them by context, while FixMiner (Koyuncu et al., 2020) mines repair patterns from edit scripts enriched with AST context. Agent systems instead distill reusable knowledge, workflows, or executable skills from prior trajectories. ExpeL (Zhao et al., 2024) and ReasoningBank (Ouyang et al., 2026) extract reusable textual knowledge, Agent Workflow Memory (Wang et al., 2025) induces reusable workflows, Voyager stores executable code skills (Wang et al., 2024), and Skill-Pro (Mi et al., 2026) filters skill updates with a surrogate objective.

Experience-guided autoformalization and repair. More directly, historical corrections have been reused to guide subsequent revisions. LTRAG retrieves translation and refinement examples (Hu et al., 2025), while PrefRAG retrieves program preferences and explanations to guide formalization refinement (Zhou & Liu, 2026). ExpeRepair combines concrete demonstrations with abstract repair insights for program repair (Mu et al., 2026). Recent systems also represent corrections as structured repair memories. MechMem-RTL reuses verifier-confirmed repair records (Cheng

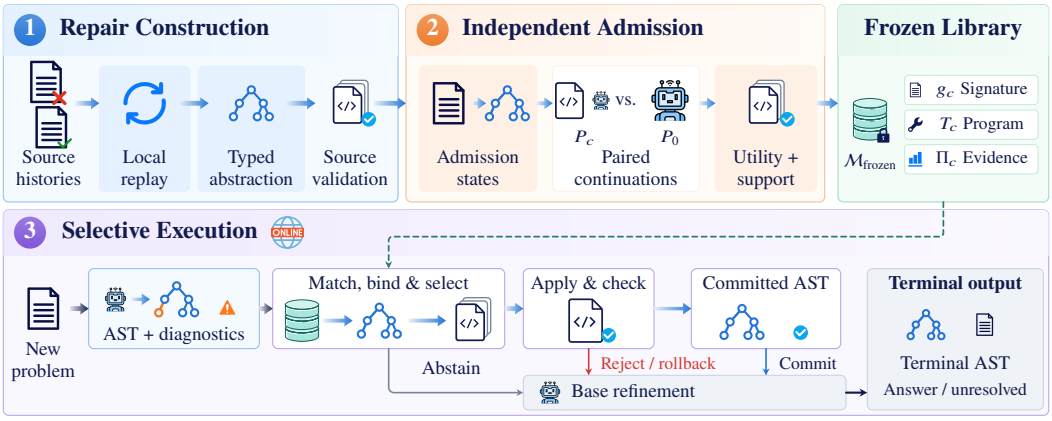


Figure 2: Overview of the core workflow of the REEXP framework.

et al., 2026), while MIRA decomposes Text-to-SQL corrections into reusable repair items checked against database evidence (Liu et al., 2026). REEXP instead derives executable local repairs directly from autoformalization correction histories without repair-type supervision, using replay, typed abstraction, and held-out admission before selective execution on new formalizations.

3 METHODOLOGY

Figure 2 summarizes the REEXP workflow. Offline, replay and typed abstraction convert failure-to-success correction episodes into reusable repair programs, and independent admission retains only those with a positive mean paired gain over base refinement on held-out states. Online, an admitted program is rebound and executed at an eligible refinement state before base refinement resumes.

3.1 PROBLEM FORMULATION

For a natural-language problem x, a formalizer F_θ produces an abstract syntax tree (AST) $z_0 = F_\theta(x)$. A symbolic interface compiles an AST z into Z3 constraints (de Moura & Bjørner, 2008) and executes them, returning runtime diagnostics and, when evaluation resolves, a task answer $\mathrm{Eval}(x, z)$. At refinement round t, the base refiner R_θ maps the state $s_t = (x, z_t, f_t)$ to a candidate AST $z_{t+1} = R_\theta(s_t)$, where f_t denotes the diagnostics observed at that state.

Given a reference answer y, we define answer correctness as $J_{\text{ans}}(x, z, y) = \mathbf{1}[\mathrm{Eval}(x, z) = y]$; invalid or unresolved evaluations score zero. Offline supervision consists only of task answers, without reference formalizations or edit-level annotations. We aim to identify local AST transformations whose reuse on compatible refinement states improves terminal accuracy over base refinement.

3.2 REPLAY-BASED REPAIR CONSTRUCTION

Correction episodes and replay reduction. Premises and declarations form the writable AST region, whereas queries, answer mappings, and evaluator definitions remain fixed. From base-refiner trajectories in an extraction set $\mathcal{D}_{\text{ext}}$, we extract failure-to-success episodes $e = (s_i, s_j)$ with $i < j$, parseable z_i and z_j, $J_{\text{ans}}(x, z_i, y) = 0$, and $J_{\text{ans}}(x, z_j, y) = 1$. The source state s_i must also provide observable diagnostics. Let $z^- = z_i$ and $z^+ = z_j$.

A deterministic AST differencing procedure over the canonical serialization yields a unique edit script $D_e = (\delta_1, \ldots, \delta_K)$ from z^- to z^+, where each atomic edit specifies an operation, location, precondition, and payload. A complete correction script may contain edits unnecessary for source-answer recovery. Replay tests order-preserving subsequences of D_e within the writable region in increasing order of length. Let $\mathcal{I}_e$ denote the tested index subsequences and T_I their edit programs. A subsequence is *source-recovering* ($\mathrm{Src}(I, e) = 1$) when its replay is valid and yields a unique resolved outcome satisfying $J_{\text{ans}}(x, T_I(z^-), y) = 1$. The retained index subsequences are $\mathcal{I}_e^* = \arg\min_{I \in \mathcal{I}_e : \mathrm{Src}(I,e)=1} |I|$. Each retained subsequence defines a source-recovering program; an episode contributes no program if no tested subsequence qualifies.

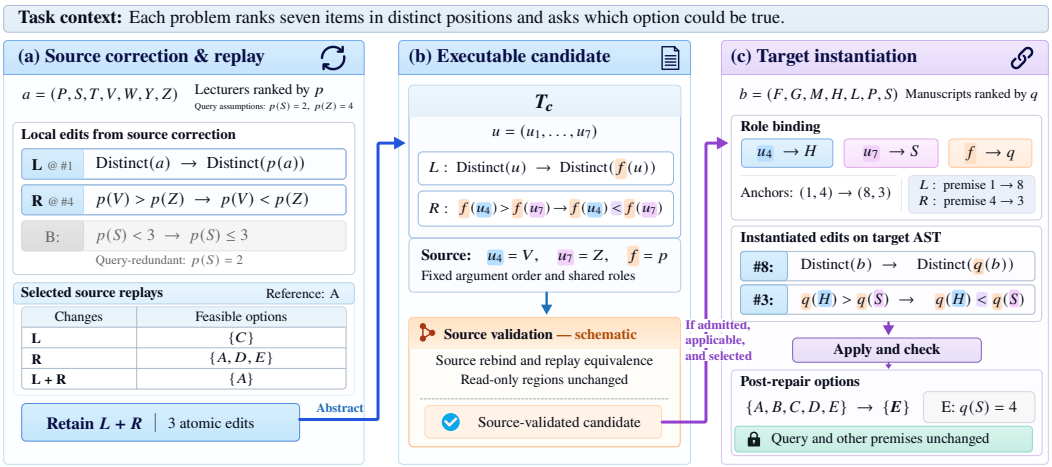


Figure 3: Worked example on two published AR-LSAT problems (Zhong et al., 2022). Blue, pink, and orange mark the shared roles u_4, u_7, and f; purple marks the rewritten operator. Appendix B provides the replay outcomes and role bindings.

In Figure 3(a), the retained program $L + R$ recovers the source answer, while the omitted edit B is redundant under the query assumption $p(S) = 2$. Here, L is a single argument-lifting edit, whereas R consists of deleting the `gt` entry and inserting the `lt` entry.

Typed abstraction. For each $I^* \in \mathcal{I}_e^*$, typed abstraction constructs a typed program T_c by replacing instance-specific symbols with typed roles and absolute edit locations with anchor templates. The same typed roles are shared across edit templates, payloads, and edit preconditions. During rebinding, entity and function symbols and anchor positions may change, while operator patterns, argument structure, arity, and variable scope remain fixed. Multiple concrete edits sharing an abstract template are collapsed into one template and expanded again at matching target sites during instantiation. Templates retaining more than $r_{\max}$ edits are discarded, where $r_{\max}$ is a fixed bound.

The resulting candidate is represented as $c = (g_c, T_c, \Pi_c)$, where g_c records the language, diagnostic class and code, and enclosing AST context; T_c contains the typed roles, anchor templates, and edit preconditions; and Π_c stores source histories and replay evidence. Figure 3(b) shows the shared roles used to reconstruct the source edits.

Source replay validation. A typed program is source-valid only if it reproduces the update selected by replay reduction. Let $\beta(T, s)$ denote the binding of the typed roles and anchors of T to state s, subject to the conditions in Section 3.4. Writing $s^- = s_i$, source replay validation first requires a successful binding $\beta(T_c, s^-) \neq \bot$. The instantiated program must then satisfy

$$T_c(z^-; \beta(T_c, s^-)) \equiv_{\text{AST}} T_{I^*}(z^-), \tag{1}$$

and produce a nonempty update that preserves the read-only region and retains source-answer recovery. Here, $\equiv_{\text{AST}}$ denotes equality under a fixed canonicalization that normalizes premise order and designated commutative structures while preserving ordered relation arguments and variable scopes.

Candidates passing source replay validation are deduplicated by signature g_c and typed program T_c to form $\mathcal{C}_{\text{src}}$; each resulting equivalence class is a *repair family*. Merged candidates pool their source histories and replay evidence in Π_c.

3.3 Independent Admission and Library Construction

Source replay establishes recovery on the source problem but does not measure reuse utility. Independent admission uses a set $\mathcal{D}_{\text{adm}}$ whose problem groups—questions sharing a background context—are disjoint from those of the extraction set $\mathcal{D}_{\text{ext}}$.

Each group in $\mathcal{D}_{\text{adm}}$ contributes its first eligible base-refiner state under the same eligibility rule used at test time. The state is fixed before candidate matching and without reference labels or

continuation outcomes. For candidate c, $\mathcal{H}_c$ contains eligible states with compatible signatures and valid bindings. Since each group contributes at most one state, $n_c = |\mathcal{H}_c|$ counts admission groups.

Two continuations are compared from each $s \in \mathcal{H}_c$ under identical refinement conditions. The repair-assisted continuation P_c attempts c once and then returns to base refinement, whereas P_0 uses base refinement throughout. Let $P_c(s)$ and $P_0(s)$ denote the terminal ASTs. The paired gain and mean admission utility are

$$\Delta_c(s) = J_{\text{ans}}(x, P_c(s), y) - J_{\text{ans}}(x, P_0(s), y), \tag{2}$$

$$\widehat{\mu}_c = \frac{1}{n_c} \sum_{s \in \mathcal{H}_c} \Delta_c(s), \qquad n_c > 0. \tag{3}$$

Here, $\Delta_c(s) \in \{-1, 0, 1\}$ indicates whether terminal correctness decreases, remains unchanged, or increases. All states in $\mathcal{H}_c$ contribute to the mean, including states where the repair is rejected and states whose terminal evaluation does not resolve.

Admission requires support from at least $n_{\min}$ admission groups and mean utility above τ:

$$\mathcal{M}_{\text{frozen}} = \left\{ c \in \mathcal{C}_{\text{src}} : n_c \geq n_{\min},\ \widehat{\mu}_c > \tau \right\}, \qquad n_{\min} \geq 1, \quad \tau \geq 0. \tag{4}$$

The estimate $\widehat{\mu}_c$ characterizes the empirical utility of c on its admission states; test-time execution is further conditioned on applicability and competition among admitted programs. The admission hyperparameters $(n_{\min}, \tau)$ are selected on a set $\mathcal{D}_{\text{sel}}$ whose problem groups are disjoint from those of $\mathcal{D}_{\text{ext}}$ and $\mathcal{D}_{\text{adm}}$, according to terminal symbolic accuracy before CoT fallback. Admitted candidates are re-indexed by m; each retains its source evidence and admission measurements in Π_m.

3.4 Selective Execution on New Problems

Eligibility and binding. A state is *eligible* for library lookup when its AST is parseable and an observable diagnostic is available. Each problem makes one library decision at its first eligible base-refiner state s_t, identified before candidate matching and without reference labels.

The binding operator $\beta(T_m, s_t)$ instantiates the typed roles and anchors of T_m subject to its edit preconditions. When a template matches multiple sites within the region indicated by the diagnostic, the corresponding edits are expanded deterministically across those sites. Binding succeeds only when all valid matching witnesses induce the same update under $\equiv_{\text{AST}}$; otherwise β returns $\perp$. Figure 3(c) shows the bound target edits and the feasible options immediately after their application.

With g_t denoting the signature extracted from (f_t, z_t), the applicable set is

$$\mathcal{M}_a(s_t) = \{ m \in \mathcal{M}_{\text{frozen}} : \text{Compatible}(g_m, g_t),\ \beta(T_m, s_t) \neq \perp \}. \tag{5}$$

Signature compatibility requires agreement in language, diagnostic class and code, and enclosing AST context, while allowing instance-specific symbols to differ.

Selection and transactional execution. For an applicable record m, let $\widehat{\mu}_m$ denote its admission utility and u_m the number of distinct extraction groups represented in its source evidence. The lexicographic selection rule

$$m_t^* = \arg \max_{m \in \mathcal{M}_a(s_t)} (\widehat{\mu}_m,\ u_m,\ -\text{id}(m)) \tag{6}$$

prefers higher admission utility, then greater extraction-group support, and finally the smaller identifier to break remaining ties deterministically. Execution yields one of three outcomes:

$$\text{Exec}(s_t) = \begin{cases} \text{abstain}, & \mathcal{M}_a(s_t) = \varnothing, \\ \text{commit}, & \text{the selected update passes the transaction checks}, \\ \text{reject}, & \text{otherwise}. \end{cases} \tag{7}$$

The transaction checks require a valid, compilable update that satisfies the edit and binding constraints, preserves read-only and untouched regions, and executes symbolically without error. Abstention leaves the AST unchanged, rejection restores the prior AST, and commitment retains the checked update. No further library lookup occurs after any outcome.

A committed repair is an intermediate update rather than a terminal prediction. Base refinement therefore resumes from the updated AST under the standard stopping policy; we refer to this as *post-commit continuation*.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate REEXP on five logical reasoning benchmarks. The evaluation sets contain 231 analytical reasoning questions from the AR-LSAT (Zhong et al., 2022) development split, 204 first-order entailment questions from FOLIO (Han et al., 2024), and 300 object-ordering questions from BIG-bench Logical Deduction (Srivastava et al., 2023); for these three, offline data are drawn from the corresponding training splits. For ProverQA (Qi et al., 2025), evaluation uses all 500 questions in the *Hard* subset, with offline data drawn from the released training split. For ZebraLogic (Lin et al., 2025), evaluation uses all 200 puzzles in the *X-Large* category, with offline data drawn from the disjoint grid-mode *Medium* category. Appendix A specifies the data partitions.

Models and Baselines. We evaluate four LLM backbones: GPT-4o (OpenAI et al., 2024), Claude Haiku 4.5 (Anthropic, 2025), DeepSeek-V3.2 in non-thinking mode (DeepSeek-AI, 2025), and Qwen3.8-27B (Qwen Team, 2026). We compare five baselines. *Direct Prompting* produces answers directly, while *CoT Prompting* (Wei et al., 2022) generates intermediate reasoning from shared demonstrations. *Logic-LM* (Pan et al., 2023) combines autoformalization with solver-feedback refinement. *LTRAG* (Hu et al., 2025) retrieves translation and correction examples, whereas *PrefRAG* (Zhou & Liu, 2026) guides refinement with program preferences and error explanations.

Evaluation Metrics. We report end-to-end accuracy over all evaluation instances, requiring an exact match of the complete grid for ZebraLogic. When a symbolic method fails to return a task answer, methods with the same backbone use the same CoT fallback answer. Repair-specific analyses report pre-fallback symbolic accuracy. For paired comparisons, we report 95% bootstrap confidence intervals with background groups as the resampling unit.

Implementation Details. We build the formalizer–solver–refiner pipeline on Logic-LM (Pan et al., 2023), using a typed JSON representation parsed into an AST and compiled to Z3. Each problem uses one initial formalization call followed by at most 3 refinement rounds, with at most 8,192 output tokens per model call. REEXP templates contain at most $r_{\max} = 3$ edits.

A separate library is constructed for each backbone–dataset pair from its corresponding extraction data. Logic-LM uses the same REEXP pipeline with library reuse disabled. Other baselines follow their released end-to-end pipelines, while controlled ablations use shared initial states. A single pair of admission hyperparameters $(n_{\min}, \tau)$ is shared across all backbone–dataset pairs and selected on $\mathcal{D}_{\text{sel}}$ by macro-averaged pre-fallback symbolic accuracy. This yields $(n_{\min}, \tau) = (2, 0.10)$, after which all libraries and admission settings are frozen before testing.

4.2 MAIN RESULTS

Table 1 reports end-to-end accuracy for all five baselines and REEXP under each backbone.

REEXP improves over Logic-LM in every backbone–dataset pair, with five-task mean gains of 6.78–14.67 points across backbones. The 95% paired bootstrap intervals exclude zero for 16 of the 20 gains. The improvement also persists where Logic-LM underperforms CoT prompting: on ProverQA with Qwen3.8-27B, Logic-LM achieves 64.20%, compared with 74.8% for CoT and 81.60% for REEXP. Because Logic-LM uses the same base pipeline with repair reuse disabled, this comparison isolates the additional contribution of cross-instance repair reuse. The gains indicate that correction histories provide reusable local information beyond repeated refinement of the current instance.

REEXP also outperforms the experience-guided baselines in most available comparisons. It exceeds LTRAG in all eight comparisons and PrefRAG in 14 of 16, tying PrefRAG on Logical Deduction with Qwen3.8-27B and trailing it only on AR-LSAT with GPT-4o (60.61% versus 61.47%). Across the 16 PrefRAG comparisons, the mean margin is 1.77 points (95% CI $[0.44, 3.09]$). LTRAG and PrefRAG retrieve prior experience to guide model-generated revisions, whereas REEXP rebinds and executes an admitted local repair before refinement resumes. This comparison highlights a distinct form of experience reuse in which the corrective operation itself is transferred across instances.

Model	Method	AR-LSAT	FOLIO	Deduction	ProverQA	ZebraLogic
GPT-4o	Direct Prompting	31.17	67.65	74.67	42.80	0.50
	CoT Prompting	32.47	72.06	88.67	48.40	1.00
	Logic-LM	35.06	<u>78.92</u>	91.33	61.40	<u>64.00</u>
	LTRAG	56.71	75.49	–	–	–
	PrefRAG	**61.47**	77.94	<u>99.00</u>	<u>78.00</u>	–
	REEXP (Ours)	<u>60.61</u>	**81.37**	**99.67**	**79.40**	**83.00**
Claude Haiku 4.5	Direct Prompting	45.89	76.47	91.00	55.80	5.00
	CoT Prompting	71.00	83.33	97.33	70.80	10.00
	Logic-LM	66.23	84.31	96.67	73.60	<u>80.00</u>
	LTRAG	76.62	85.29	–	–	–
	PrefRAG	<u>81.82</u>	<u>85.78</u>	<u>98.67</u>	<u>78.40</u>	–
	REEXP (Ours)	**82.25**	**86.27**	**100.00**	**81.20**	**88.50**
DeepSeek V3.2	Direct Prompting	34.63	68.14	70.00	38.60	0.50
	CoT Prompting	80.09	<u>82.84</u>	**100.00**	63.40	2.00
	Logic-LM	72.73	80.39	96.67	66.00	<u>75.00</u>
	LTRAG	84.42	82.35	–	–	–
	PrefRAG	<u>88.74</u>	<u>82.84</u>	<u>99.33</u>	<u>76.80</u>	–
	REEXP (Ours)	**90.04**	**85.29**	**100.00**	**78.40**	**92.50**
Qwen3.8-27B	Direct Prompting	74.89	77.94	83.33	72.80	60.00
	CoT Prompting	93.07	<u>85.29</u>	<u>99.67</u>	74.80	52.50
	Logic-LM	93.51	81.86	96.33	64.20	<u>92.50</u>
	LTRAG	93.07	77.94	–	–	–
	PrefRAG	<u>94.81</u>	82.35	**100.00**	<u>76.00</u>	–
	REEXP (Ours)	**96.97**	**87.25**	**100.00**	**81.60**	**96.50**

Table 1: End-to-end accuracy (%). Within each backbone, **bold** and <u>underlined</u> values indicate the best and second-best results, respectively; tied values share the same mark. A dash (–) indicates that the method provides no released implementation for that benchmark.

Across the full table, REEXP achieves the highest or joint-highest accuracy among methods with reported results in 19 of the 20 backbone–dataset pairs. The gain magnitude does not simply follow baseline headroom: on Logical Deduction, REEXP improves Logic-LM from 91.33–96.67% to 99.67–100.00%, whereas FOLIO starts from the lower range of 78.92–84.31% but yields a mean gain of only 3.68 points. This contrast points to the availability of recurring, transferable correction patterns as an additional factor shaping the benefit of reuse. The subsequent analyses examine these patterns through component ablations, repair-family composition, and observed reuse behavior.

4.3 ABLATION STUDY

Table 2 examines the contributions of offline repair construction, independent admission, and online execution. Removing replay reduction lowers accuracy by 1.30–3.00 percentage points across the six backbone–dataset pairs, while removing typed abstraction causes larger drops of 2.16–7.00 points. Despite retaining larger source-valid candidate pools, both variants perform worse than the full method. The larger degradation without typed abstraction highlights the importance of rebindable repair structure over the sheer volume of extracted source corrections. Removing independent admission decreases accuracy by 0.50–2.45 points. Although these candidates recover their source problems, retaining them without held-out utility validation weakens downstream performance, distinguishing source recovery from reuse utility.

The online ablations hold the frozen libraries fixed. Replacing direct execution with model-generated revision from the selected repair reduces accuracy by 0.43–1.50 points, showing a consistent benefit from applying the rebound repair directly to the symbolic state. Removing post-commit continuation causes the larger reduction of 1.73–5.00 points. Together, these results indicate that repair execution and subsequent refinement play complementary roles: the repair supplies a local structural update, while continued refinement completes the remaining reasoning process.

Stage	Variant	Claude Haiku 4.5				Qwen3.8-27B			
		#Exp.	AR-LSAT	FOLIO	ZebraLogic	#Exp.	AR-LSAT	FOLIO	ZebraLogic
REEXP (Full)		34	**82.25**	**86.27**	**88.50**	26	**96.97**	**87.25**	**96.50**
Offline	w/o replay reduction	42	80.95	84.80	86.00	43	95.67	85.29	93.50
Offline	w/o typed abstraction	68	79.22	82.35	82.00	89	94.81	83.33	89.50
Offline	w/o admission	34	80.09	83.82	87.00	26	95.24	85.29	96.00
Online	w/o direct execution	🔒	81.39	85.29	87.00	🔒	96.54	86.27	95.50
Online	w/o continuation	🔒	80.52	83.82	85.00	🔒	95.24	84.31	91.50

Table 2: Offline and online ablations on AR-LSAT, FOLIO, and ZebraLogic. Accuracy is reported in %. #Exp. denotes deduplicated source-valid repair families summed over the three datasets; 🔒 denotes the frozen library of the full method.

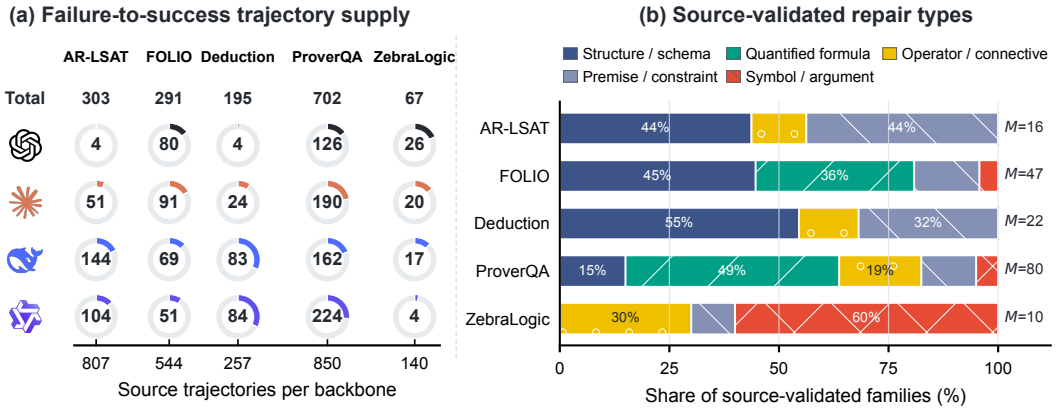


Figure 4: Supply and composition of repair experience. (a) Failure-to-success recovery rates across benchmarks and backbones, with source-trajectory counts shown below each benchmark and totals aggregated over backbones. (b) Distribution of source-validated repair families by repair type; M denotes distinct families after deduplication across backbones.

4.4 Origin and Composition of Repair Experience

Figure 4 examines whether successful refinement trajectories contain recurring repair structure and how that structure varies across benchmarks. Across the five benchmarks, 1,558 failure-to-success trajectories yield 168 distinct source-validated repair families, showing that many successful corrections instantiate recurring local patterns. Their supply is not determined by trajectory volume alone: AR-LSAT and ProverQA have similar source-trajectory counts per backbone (807 and 850) but yield 303 and 702 recoveries, while AR-LSAT and FOLIO produce similar numbers of recoveries (303 and 291) yet only 16 and 47 repair families. Successful trajectories therefore differ substantially in the reusable structure they expose.

Panel (b) further shows that repair categories recur across benchmarks while individual families remain largely benchmark-specific. Quantified-formula repairs concentrate in the first-order reasoning tasks, whereas ZebraLogic is dominated by symbol/argument edits. Across datasets, 175 family memberships correspond to 168 unique families, with only seven shared by two benchmarks. The repair space thus exhibits shared structural categories but task-specific executable patterns, consistent with typed abstraction over separately constructed repair libraries.

4.5 Repeated Cross-Instance Repair Reuse

Figure 5 characterizes repeated use of source-validated repair families across distinct target backgrounds. A use denotes a committed application, with each family–background pair counted once across backbones. Across the five benchmarks, 50.0–80.0% of source families are used at least once, and 64.3–87.5% of those recur on a second target background. Most used families therefore support repeated cross-instance reuse rather than a single successful transfer.

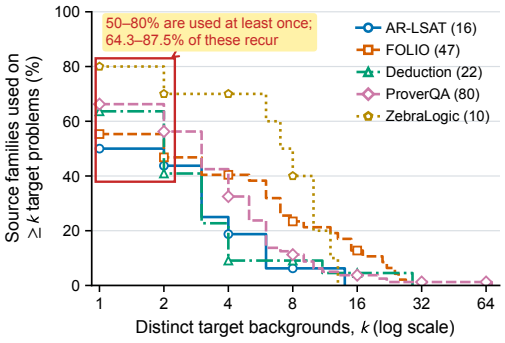


Figure 5: Repeated cross-instance reuse of source-validated repair families. Each curve shows the fraction of families committed on at least k distinct target backgrounds, counting each family–background pair once across backbones.

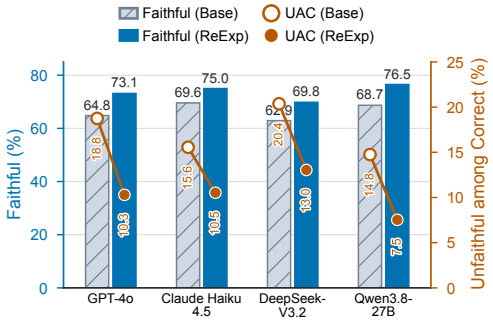


Figure 6: Semantic faithfulness of paired ProverQA continuations. Bars denote the proportion of faithful formalizations; markers denote unfaithfulness among answer-correct representations (UAC).

Recurrence varies substantially across benchmarks. AR-LSAT and Logical Deduction show a rapid decline after the first few targets, whereas FOLIO retains more families at intermediate recurrence levels and ProverQA exhibits the longest tail. ZebraLogic shows the highest rate of initial reuse but substantially less persistence at larger recurrence counts. Overall, reuse is concentrated in a subset of repair families, consistent with the selective execution policy of REEXP. Some repairs recur across many target instances, whereas others remain narrowly applicable.

4.6 SEMANTIC FAITHFULNESS

A correct answer does not guarantee a faithful formalization. We audit paired ProverQA continuations from states at which REEXP commits a repair and both REEXP and base refinement produce terminal symbolic representations available for semantic annotation. This yields 451 matched pairs across the four backbones. The paired continuations start from the same state with identical remaining budgets and differ only in the repair decision. Two annotators with training in formal logic independently assess the terminal representations before CoT fallback, blinded to method identity and answer correctness (Cohen's $\kappa = 0.93$).

Figure 6 shows consistent improvements across all four backbones. Relative to base refinement, REEXP increases the faithful share by 5.4–8.3 percentage points and reduces unfaithfulness among answer-correct representations (UAC) by 5.0–8.5 points. Both measures improve for every backbone: REEXP produces more semantically faithful terminal formalizations while reducing semantic errors among otherwise correct answers. These results show that the accuracy gains from repair reuse are accompanied by improved semantic quality in the resulting formalizations.

5 CONCLUSION

In this paper, we introduced REEXP, a framework for reusing autoformalization corrections. REEXP converts observed corrections into rebindable repair programs, validates their utility on held-out states, and selectively executes them during subsequent refinement. Across five logical reasoning benchmarks and four backbones, REEXP consistently improves over within-instance solver-feedback refinement and outperforms retrieval-based experience reuse in most comparisons. The repair analysis further shows that successful corrections contain recurring local structures reusable across new targets, while accuracy gains are accompanied by more faithful formalizations.

The current library construction depends on observed failure-to-success correction histories, so the coverage and diversity of reusable repairs are constrained by the experience available during offline construction. Nevertheless, the recurring repair structures identified across benchmarks suggest that correction histories provide a promising substrate for experience reuse in autoformalization. Future work can explore more active and adaptive library construction, including the discovery, expansion, and composition of repair experience beyond passively observed corrections.

AI USE STATEMENT

We used generative AI tools to assist with the preparation and presentation of experimental analysis figures and with manuscript editing, including language polishing, formatting, and typesetting. The authors reviewed all AI-assisted work and take responsibility for the final content of this paper, including its text, claims, and artifacts.

REFERENCES

Daneshvar Amrollahi, Jerry Lopez, and Clark Barrett. Faithful autoformalization via roundtrip verification and repair. *arXiv preprint arXiv:2604.25031*, 2026.

Anthropic. System card: Claude Haiku 4.5. System card, October 2025. URL `https://www.anthropic.com/claude-haiku-4-5-system-card`.

Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. Getafix: Learning to fix bugs automatically. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):159:1–159:27, 2019.

Mingyu Cheng, Junjie Gao, Jinhua Cui, and Kuncai Zhong. MechMem-RTL: Reusing verified mechanism memories for LLM-based RTL repair. *arXiv preprint arXiv:2607.17053*, 2026.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

DeepSeek-AI. DeepSeek-V3.2 model card. Hugging Face model card, 2025. URL `https://huggingface.co/deepseek-ai/DeepSeek-V3.2`.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander Fabbri, Wojciech Maciej Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22017–22031, 2024.

Ruikang Hu, Shaoyu Lin, Yeliang Xiu, and Yongmei Liu. LTRAG: Enhancing autoformalization and self-refinement for logical reasoning with thought-guided RAG. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 2483–2493, 2025.

Shashank Kirtania, Priyanshu Gupta, and Arjun Radhakrishna. LOGIC-LM++: Multi-step refinement for symbolic formulations. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pp. 56–63, 2024.

Anil Koyuncu, Kui Liu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, Martin Monperrus, and Yves Le Traon. FixMiner: Mining relevant fix patterns for automated program repair. *Empirical Software Engineering*, 25(3):1980–2024, 2020.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. ZebraLogic: On the scaling limits of LLMs for logical reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 37889–37905, 2025.

Yining Liu, Chenyu Yang, Boyan Li, Rui Mao, and Yuyu Luo. MIRA: Evidence-verified repair memory for text-to-SQL correction. *arXiv preprint arXiv:2608.06950*, 2026.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-Pro: Learning reusable skills from experience via non-parametric PPO for LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. ExpeRepair: Dual-memory enhanced LLM-based repository-level program repair. *Proceedings of the ACM on Software Engineering*, 3(FSE):3954–3976, 2026.

Zhiyu Ni, Zheng Liang, Liangcheng Song, Chenrui Cao, Xian Zhang, Alberto Sangiovanni-Vincentelli, and Pierluigi Nuzzo. Draft-and-Prune: Improving the reliability of auto-formalization for logical reasoning. *arXiv preprint arXiv:2603.17233*, 2026.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5153–5176, 2023.

OpenAI et al. GPT-4o system card, 2024. URL `https://arxiv.org/abs/2410.21276`.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *Proceedings of the International Conference on Learning Representations*, 2026.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, 2023.

Rizky Ramadhana Putra, Raihan Sultan Pasha Basuki, Yutong Cheng, and Peng Gao. NL2Logic: AST-guided translation of natural language into first-order logic with large language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6035–6051, 2026.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. Large language models meet symbolic provers for logical reasoning evaluation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Qwen Team. Qwen3.8-27B model card. Hugging Face model card, 2026. URL `https://huggingface.co/Qwen/Qwen3.8-27B`. Accessed September 24, 2026.

Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Björn Hartmann. Learning syntactic program transformations from examples. In *Proceedings of the 39th International Conference on Software Engineering*, pp. 404–415, 2017.

Hyun Ryu, Gyeongman Kim, Hyemin S. Lee, and Eunho Yang. Divide and translate: Compositional first-order logic translation and verification for complex logical reasoning. In *Proceedings of the International Conference on Learning Representations*, 2025.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 63897–63911, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. SatLM: Satisfiability-aided language models using declarative prompting. In *Proceedings of the 37th Conference on Neural Information Processing Systems*, 2023.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Yining Chen, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. Analytical reasoning of text. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 2306–2319, 2022.

Yuyin Zhou and Yongmei Liu. PrefRAG: Correcting semantic errors in auto-formalization for logical reasoning with program preference RAG. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16193–16208, 2026.

A Data Partitions and Split Provenance

Each benchmark is partitioned into four disjoint data roles. $\mathcal{D}_{\mathrm{ext}}$ supplies correction trajectories for candidate extraction, $\mathcal{D}_{\mathrm{adm}}$ provides paired comparisons for independent admission, $\mathcal{D}_{\mathrm{sel}}$ selects the admission thresholds $(n_{\min}, \tau)$, and $\mathcal{D}_{\mathrm{test}}$ evaluates the frozen library. Partitioning uses background contexts rather than individual instances: instances sharing a source identifier or matching after context normalization remain in the same partition. Disjointness therefore holds at the background-context level. Table 3 reports instance and background-context counts.

For AR-LSAT, FOLIO, and Logical Deduction, the three offline partitions are drawn from the training splits used by Logic-LM (Pan et al., 2023), while $\mathcal{D}_{\mathrm{test}}$ uses the corresponding development splits with 231, 204, and 300 evaluation instances, respectively. For ProverQA, the offline partitions use the released training split and $\mathcal{D}_{\mathrm{test}}$ contains the 500 instances in the *Hard* subset. For ZebraLogic, the offline partitions use the grid-mode *Medium* category and $\mathcal{D}_{\mathrm{test}}$ contains 200 *X-Large* puzzles; the two sets also differ in grid size.

Dataset	$\boldsymbol{\mathcal{D}_{\mathrm{ext}}}$	$\boldsymbol{\mathcal{D}_{\mathrm{adm}}}$	$\boldsymbol{\mathcal{D}_{\mathrm{sel}}}$	$\boldsymbol{\mathcal{D}_{\mathrm{test}}}$
AR-LSAT	807 (135)	516 (87)	262 (46)	231 (40)
FOLIO	544 (186)	347 (116)	113 (38)	204 (73)
Deduction	257 (57)	150 (34)	85 (17)	300 (192)
ProverQA	850 (850)	650 (620)	400 (400)	500 (500)
ZebraLogic	140 (140)	84 (84)	56 (56)	200 (200)

Table 3: Data partitions for the five benchmarks, reported as instances (background contexts) for extraction, independent admission, threshold selection, and testing.

B Repair Catalog and Worked Example

B.1 Catalog Composition

Figure 7 summarizes source-catalog construction across the five benchmarks, pooled over four backbones. All counts are measured before independent admission into the deployment library. Recovered trajectories begin with an incorrect symbolic answer and end with a correct one before CoT fallback. Eligible segments additionally satisfy the extraction conditions in Section 3.2, while source-validated program occurrences pass replay and source-binding validation.

The construction stages use different counting units: a trajectory may contain multiple eligible segments, and multiple program occurrences may collapse into one repair family. Across the five benchmarks, catalog construction yields 1,558 recovered trajectories, 926 eligible segments, and 280 source-validated program occurrences, which form 175 dataset-level family memberships.

Families are assigned to five structural categories according to their stored edits and anchor patterns. For families matching multiple categories, the precedence is structure/schema, quantified formula, operator/connective, premise/constraint, and symbol/argument. The categories characterize the structure being modified. After deduplication across datasets, the five categories contain 47, 55, 23, 31, and 12 unique families, respectively. Among the 168 families, 62 (36.9%) contain one edit, 84 (50.0%) contain two, and 22 (13.1%) contain three.

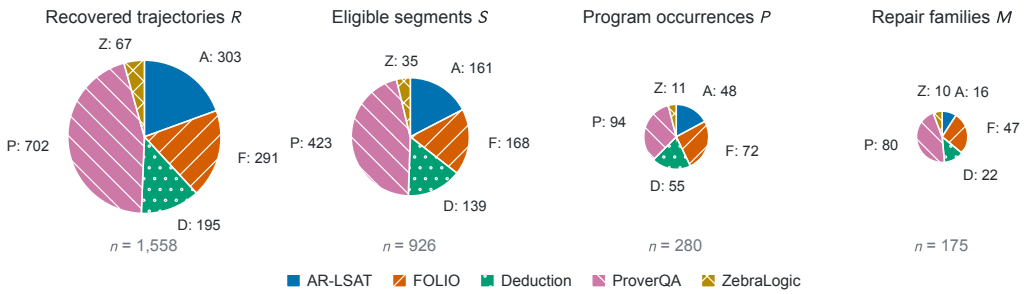


Figure 7: Source-catalog construction across five benchmarks, pooled over four backbones. Family counts are deduplicated within datasets; 175 memberships correspond to 168 unique families.

Category	Diagnostic (g_c)	Anchor and update (T_c)
Structure/schema	Unknown symbol	sort declaration missing a term $\mathcal{C}(S) \longrightarrow \mathcal{C}(S) \cup \{c\}$
Quantified formula	Conflicting premises	universal body with two unary predicates $\forall x{:}S\,[P(x) \wedge Q(x)] \longrightarrow \forall x{:}S\,[P(x) \Rightarrow Q(x)]$
Operator/connective	Conflicting premises	adjacency in a seven-child conjunction $p+1=q \longrightarrow (p+1=q) \vee (q+1=p)$
Premise/constraint	Underdetermined	premise list with three calls to one function append $\bigwedge_{i=1}^{3}(1 \le f(u_i) \wedge f(u_i) \le 3)$
Symbol/argument	Conflicting premises	binary disequality of integer variables $v_i \neq v_j \longrightarrow v_i \neq 2$

Table 4: Representative source-validated family from each repair category. Rows are drawn from FOLIO, ProverQA, AR-LSAT, Logical Deduction, and ZebraLogic, respectively. Symbols are rebound roles; numeric literals and anchor structures remain fixed.

Table 4 presents one representative family from each category. Signature compatibility restricts candidate families, while anchor templates constrain their valid bindings. Three examples share the diagnostic *conflicting premises* but require different anchor structures and local updates, showing that the diagnostic alone does not determine a repair; the typed context and anchors specify the transformation that can be rebound to a compatible target.

B.2 Source Support of Repair Families

A stored family $c = (g_c, T_c, \Pi_c)$ combines a compatibility signature, a typed repair program, and its source evidence. The signature g_c restricts compatible target states, T_c specifies the rebindable roles, anchors, preconditions, and local update, and Π_c retains the source episodes and validation evidence supporting the family.

Source support is sparse across the catalog. The median family is supported by one source instance, with a mean support of 1.63; 140 of the 168 families have a single source instance, while only one is supported by 19. Provenance is similarly concentrated: 164 families originate from a single backbone and 161 from a single dataset. These statistics show that source recovery is typically established from limited evidence, motivating the independent measurement of reuse utility on $\mathcal{D}_{\mathrm{adm}}$.

B.3 Replay Reduction and Target Rebinding

The example in Figure 3 uses constructed faulty formalizations of two published AR-LSAT problems (Zhong et al., 2022; Pan et al., 2023); both backgrounds are excluded from all reported measurements. Table 5 compares the source and target role bindings and the symbolic outcomes immediately after each local update, before subsequent refinement or CoT fallback.

Neither L nor R alone recovers source answer A, whereas $L + R$ does. The lifting edit L constrains the distinctness of ranks, which does not follow from the distinctness of the enumerated entities, while the relation rewrite R corrects the ordering constraint. Of the five atomic edits in the full

correction, replay retains the shortest source-recovering program $L + R$, consisting of three edits: R requires a complete delete/insert pair, whereas the two edits implementing B leave the feasible assignments unchanged under the source assumption $p(S) = 2$.

The shared roles u_4, u_7, f reinstantiate the retained program on a target with different symbols, sorts, and premise locations, where the rebound update resolves the target answer. Replay therefore removes edits unnecessary for source recovery, while typed abstraction preserves the relational structure needed to rebind the retained program across instances.

Component	Source	Target
Entity roles $\mathbf{u}$	(P, S, T, V, W, Y, Z)	(F, G, M, H, L, P, S)
Function role f	$p : \text{Lecturer} \to \{1, \dots, 7\}$	$q : \text{Manuscript} \to \{1, \dots, 7\}$
Lift L	$\text{Distinct}(\mathbf{u}) \mapsto \text{Distinct}(p(\mathbf{u}))$	$\text{Distinct}(\mathbf{u}) \mapsto \text{Distinct}(q(\mathbf{u}))$
Relation R	$p(V) > p(Z) \mapsto p(V) < p(Z)$	$q(H) > q(S) \mapsto q(H) < q(S)$
Omitted B	$p(S) < 3 \mapsto p(S) \leq 3$	No corresponding edit
Locations (L, R)	Premises $(1, 4)$	Premises $(8, 3)$
Before repair	135 assignments; options A, C, D, E	195 assignments; options A–E
After L only	1 assignment; option C	—
After R only	243 assignments; options A, D, E	—
After $L + R$	2 assignments; option A	4 assignments; option E

Table 5: Source replay and target rebinding for Figure 3. Assignment counts exclude answer-option constraints; listed options remain feasible under the resulting symbolic state. L is one atomic edit, while R is a delete/insert pair.

C Additional Evaluation

C.1 Policy Execution and Outcomes

Because ReExp differs from Logic-LM only through library execution, instances without a committed repair follow the same base-refinement trajectory, allowing Table 6 to decompose the end-to-end accuracy difference into terminal improvements and degradations on committed instances. Across the 20 settings, 3,168 of 5,740 evaluation instances reach an eligible state, of which 1,880 abstain, 217 reject the proposed update, and 1,071 commit a repair. Among committed instances, 695 improve and 90 degrade relative to Logic-LM, yielding a net gain of 605 answers. The net effect is positive in every backbone–dataset setting, with degradations accounting for 8.4% of commits.

C.2 Repeated Cross-Instance Use

Figure 5 summarizes repeated use of repair families across target backgrounds. For each dataset d, $\mathcal{F}_d$ contains the source-validated families originating from that dataset: 16, 47, 22, 80, and 10 families for AR-LSAT, FOLIO, Logical Deduction, ProverQA, and ZebraLogic, respectively. A family with sources in two datasets belongs to both inventories.

A use denotes a committed application on a distinct target background. Family–background pairs are counted once across backbones. For $m \in \mathcal{F}_d$, let $k_{m,d}$ denote the number of target backgrounds in dataset d on which m commits. The reuse profile is

$$R_d(k) = \frac{100}{|\mathcal{F}_d|} \sum_{m \in \mathcal{F}_d} \mathbf{1}[k_{m,d} \geq k], \qquad k \geq 1, \tag{8}$$

where the denominator includes families that are never applied.

Table 7 gives the counts underlying these curves. *Used* and *Repeated* denote families with $k_{m,d} \geq 1$ and $k_{m,d} \geq 2$, respectively; *Applications* sums $k_{m,d}$ over the inventory, and *Maximum* is the largest observed $k_{m,d}$. Thus $R_d(1)$ measures reuse breadth, while recurrence and application counts characterize reuse depth across the families that are actually used.

Backbone	Dataset	N	Eligible	Abstain	Reject	Commit	Improve	Degrade	Net
GPT-4o	AR-LSAT	231	198	100	14	84	66	7	59
	FOLIO	204	176	147	8	21	8	3	5
	Deduction	300	64	20	6	38	28	3	25
	ProverQA	500	468	298	27	143	101	11	90
	ZebraLogic	200	75	17	6	52	43	5	38
Claude Haiku 4.5	AR-LSAT	231	142	68	13	61	42	5	37
	FOLIO	204	118	95	7	16	6	2	4
	Deduction	300	31	10	4	17	12	2	10
	ProverQA	500	402	269	25	108	45	7	38
	ZebraLogic	200	52	15	6	31	20	3	17
DeepSeek V3.2	AR-LSAT	231	113	45	11	57	45	5	40
	FOLIO	204	124	92	8	24	13	3	10
	Deduction	300	29	9	4	16	12	2	10
	ProverQA	500	431	275	25	131	71	9	62
	ZebraLogic	200	70	16	6	48	39	4	35
Qwen3.8-27B	AR-LSAT	231	41	17	5	19	10	2	8
	FOLIO	204	115	82	7	26	14	3	11
	Deduction	300	33	11	4	18	13	2	11
	ProverQA	500	447	282	27	138	97	10	87
	ZebraLogic	200	39	12	4	23	10	2	8
All 20 settings			3,168	1,880	217	1,071	695	90	605

Table 6: Frozen-policy behavior on the test sets. *Eligible* denotes instances reaching a valid library-decision state; *Abstain*, *Reject*, and *Commit* are the corresponding policy outcomes. Among committed instances, *Improve* and *Degrade* count terminal improvements and degradations relative to Logic-LM, with $\text{Net} = \text{Improve} - \text{Degrade}$.

Dataset	Families	Used	Repeated	Applications	Maximum
AR-LSAT	16	8	7	33	13
FOLIO	47	26	22	243	26
Deduction	22	14	9	60	28
ProverQA	80	53	45	298	67
ZebraLogic	10	8	7	60	12

Table 7: Repeated cross-instance use of source-validated repair families. *Used* and *Repeated* count families committed on at least one and two target backgrounds, respectively; *Applications* counts distinct family–background pairs, and *Maximum* gives the largest reuse count for one family. Pairs are deduplicated across backbones.

C.3 Online Computational Cost

We compare the online resource use of Logic-LM, LTRAG, PrefRAG, and ReExp with Qwen3.8-27B on the evaluation sets of Table 1. Logic-LM and ReExp share the formalizer, initial programs, feedback interface, and 3-round refinement budget. LTRAG and PrefRAG follow their released end-to-end pipelines; in LTRAG, retrieval also enters the translation stage. Model calls and input/output tokens are averaged over evaluation instances and include formalization, refinement, and CoT fallback; solver calls count complete symbolic evaluations, including resource use from rejected repair attempts.

Table 8 shows a distinct resource profile for executable repair reuse. Across all reported comparisons, ReExp uses fewer input tokens than the retrieval-based baselines: PrefRAG and LTRAG consume 1.5–7.9 and 1.4–2.3 times as many input tokens per instance, respectively. This difference reflects how prior experience enters inference: ReExp executes a bound repair on the symbolic state, whereas retrieval methods incorporate retrieved experience into the model context. Total generation cost remains task-dependent. Relative to Logic-LM, ReExp uses fewer model calls on FOLIO and ProverQA but more on the other three benchmarks, while output-token use increases on four of

Task	Method	Calls	Input tok.	Output tok.	Solver evals.	Acc. (%)
AR-LSAT	</> Logic-LM	3.67	7,340	4,783	2.92	93.51
	LTRAG	4.04	16,151	7,911	3.44	93.07
	PrefRAG	2.58	17,268	5,764	2.52	94.81
	REEXP	4.27	11,420	16,231	1.60	96.97
FOLIO	</> Logic-LM	2.42	4,849	1,829	2.28	81.86
	LTRAG	2.13	3,289	755	2.13	77.94
	PrefRAG	2.36	11,078	3,174	2.35	82.35
	REEXP	1.35	1,415	1,359	1.10	87.25
Deduction	</> Logic-LM	1.04	1,012	389	1.00	96.33
	PrefRAG	2.00	14,808	1,618	2.00	100.00
	REEXP	3.51	4,674	13,576	1.26	100.00
ProverQA	</> Logic-LM	2.05	4,923	2,604	2.03	64.20
	PrefRAG	2.08	20,317	3,192	2.08	76.00
	REEXP	1.52	2,561	3,627	1.29	81.60
ZebraLogic	</> Logic-LM	1.30	1,495	1,948	1.24	92.50
	REEXP	1.33	4,334	3,449	1.24	96.50

Table 8: Mean online resource use per evaluation instance with Qwen3.8-27B. Reported accuracies are those in Table 1.

the five benchmarks. Executable repair reuse therefore reduces retrieval-context overhead without uniformly reducing subsequent refinement cost.

Offline construction. Replay reduction and typed abstraction are symbolic and require no model calls. The offline model cost is concentrated in collecting and evaluating repair experience, particularly the paired continuations used for independent admission. Across the five benchmarks, the offline procedure uses 41,880 model calls and approximately 12.4M output tokens per backbone. This cost is incurred before evaluation, after which the repair library is frozen.

C.4 REFINEMENT-BUDGET SATURATION

We examine the refinement trajectory of Logic-LM with Qwen3.8-27B across the five benchmarks. Round 0 denotes the initial formalization, followed by three successive refinement rounds. At each round, we report symbolic accuracy and end-to-end accuracy with the CoT fallback used in the main evaluation. Symbolic accuracy treats unresolved outputs as incorrect, whereas the end-to-end metric applies the fallback policy of Section 4.1.

Figure 8 shows that most symbolic improvement occurs in the first refinement round. FOLIO rises from 0.0% to 67.2% and ProverQA from 1.2% to 62.2%, while the additional gain from round 1 to round 3 is below 1.5 percentage points on every benchmark and no benchmark changes between rounds 2 and 3. The three-round budget therefore captures the observed refinement plateau for Qwen3.8-27B.

End-to-end accuracy with fallback follows a different pattern. It is unchanged from round 1 through round 3 on all five benchmarks, but relative to round 0 decreases from 84.8% to 81.9% on FOLIO and from 74.6% to 64.2% on ProverQA despite large gains in symbolic accuracy. Under the evaluation policy, a resolved symbolic prediction replaces the CoT fallback, so newly resolved but incorrect formalizations can offset gains from successful refinement. This divergence motivates evaluating repair utility by paired answer correctness rather than by symbolic resolution alone.

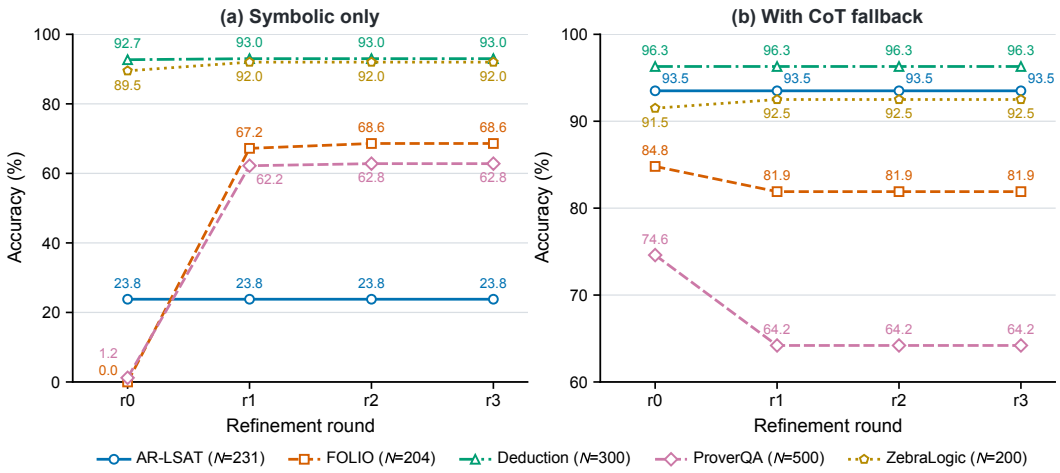


Figure 8: Logic-LM accuracy by refinement round with Qwen3.8-27B. Rounds r0–r3 denote the initial formalization and three successive refinement rounds, evaluated (a) without and (b) with CoT fallback; panel (b) uses a truncated vertical axis.

D Semantic Faithfulness Audit Details

Backbone	Method	All representations				Answer-correct subset				Paired transitions	
		n	F	U	I	C	F_C	U_C	I_C	$U \to F$	$F \to U$
GPT-4o	Base refinement	113	70	38	5	50	39	9	2	–	–
	ReExp	113	79	29	5	71	61	7	3	13	4
Claude Haiku 4.5	Base refinement	96	64	28	4	47	38	7	2	–	–
	ReExp	96	69	23	4	59	51	6	2	8	3
DeepSeek V3.2	Base refinement	122	73	43	6	56	43	11	2	–	–
	ReExp	122	81	35	6	72	60	9	3	12	4
Qwen3.8-27B	Base refinement	120	79	36	5	63	52	9	2	–	–
	ReExp	120	88	27	5	96	86	7	3	13	4

Table 9: Annotation counts underlying the ProverQA semantic-faithfulness analysis in Figure 6. F, U, and I denote faithful, unfaithful, and indeterminate representations; subscript C restricts counts to the answer-correct subset. $U \to F$ and $F \to U$ denote paired label transitions from base refinement to ReExp.

Annotation protocol. This appendix details the annotation protocol underlying the semantic-faithfulness analysis in Section 4.6. The audit uses paired terminal representations from base refinement and ReExp on ProverQA. Within each backbone, each included background contributes one matched pair obtained from the same repair-commit state under an identical remaining refinement budget. A committed state is included only when both paired continuations produce terminal symbolic representations available for semantic annotation; this criterion yields 451 matched pairs.

Two annotators with graduate-level training in first-order logic assess each representation independently using the same criteria, blinded to method identity and answer correctness. The criteria are calibrated on a disjoint set of representations from $\mathcal{D}_{\text{sel}}$ before annotation. Inter-annotator agreement is high (Cohen's $\kappa = 0.93$). A final *faithful* or *unfaithful* label requires agreement between the two annotators; disagreements and cases without a determinate judgment are recorded as *indeterminate*.

Annotation criteria. The audit evaluates each terminal symbolic representation against the semantics of the source problem. A representation is labelled *faithful* when it preserves the problem semantics under logically equivalent rewrites, and *unfaithful* when a concrete semantic mismatch is identified. Cases without a determinate judgment are labelled *indeterminate*. Solver outcomes are not used as evidence of semantic faithfulness.

Counts and metrics. Table 9 reports annotation counts by backbone and method. For all terminal representations, F, U, and I denote faithful, unfaithful, and indeterminate labels; within the C answer-correct representations, F_C, U_C, and I_C denote the corresponding counts. The audit contains 451 matched pairs, or 902 terminal representations, including 40 indeterminate cases.

The metrics in Figure 6 exclude indeterminate cases. The faithful share is $100F/(F+U)$, while unfaithfulness among answer-correct representations (UAC) is $100U_C/(F_C+U_C) = 100U_C/(C-I_C)$. Both are computed on the terminal representations produced by the paired continuations. Faithfulness and answer correctness remain distinct because a faithful audited representation may still be unresolved at evaluation.