Prove What You Can: Hybrid Neurosymbolic Reasoning for Logical Natural Language Inference

Anonymous EMNLP submission

Abstract

Logical reasoning over natural language requires both semantic interpretation and deductive search. Large language models (LLMs) can propose plausible intermediate derivations, but they often fail to compose them into a complete proof and overpredict uncertainty. Neurosymbolic methods improve search by translating an *entire* problem into first-order logic and handing it to a prover, but a single bad clause can make the whole translation non-executable. We present **Hybrid Reasoning**, a three-stage pipeline that decouples *proposing* a proof, *validating* its steps, and *deciding* a label. An LLM proposes a structured proof tree with a raw verdict; a theorem prover validates each derivation *locally*, keeping only the subset it can prove; and an *uncertainty-gated repair* attempts to re-derive the label from the premises plus the validated derivations, when the raw verdict is UNCERTAIN, so confident predictions are never overwritten. Across four benchmarks (FOLIO, ProofWriter, ProverQA, PrOntoQA) and five contemporary LLMs, Hybrid Reasoning is the top-performing method on the two hardest datasets and reaches ceiling-level accuracy on the easiest. The repair stage operates at 78-98% precision, producing 326 correct relabellings against 19 regressions; a complementarity analysis across all configurations shows a net gain of 508 predictions corrected over those that regressed. Validation depth further predicts reliability, giving Hybrid Reasoning a label-free confidence signal unavailable to global formalization pipelines.

1 Introduction

Logical reasoning over natural language requires both semantic interpretation and deductive search. A system must map linguistically varied premises into the right semantic relations, but it must also determine whether a hypothesis h follows from a set of premises P, whether its negation follows, or whether the evidence is insufficient. Large language models are flexible semantic interpreters, yet their multi-step reasoning is often not *faithful*: intermediate steps may not entail their conclusions, and the final answer can be right for the wrong reasons or wrong despite a mostly useful chain (Wei et al., 2022; Kojima et al., 2022; Saparov and He, 2023). For logical natural language inference (NLI), this is especially costly because a single unsound or missing step can flip the label.

Two families of methods address different sides of this problem. *Neurosymbolic* approaches translate the whole problem into a formal language and discharge it with a symbolic solver, inheriting the solver's deductive strength (Pan et al., 2023; Olausson et al., 2023; Ye et al., 2023). They are powerful when the translation is faithful and executable, but brittle otherwise: a single mistranslated quantifier, missing premise, or undeclared predicate can make the entire program fail. *Process-level* approaches instead score or search over individual reasoning steps (Lightman et al., 2024; Yang et al., 2022; Kamoi et al., 2026), but typically rely on learned verifiers that are themselves approximate and offer no symbolic guarantee.

Our starting observation is that proof-tree reasoning and formalization fail differently. A proof-tree model may capture useful semantic derivations but fail to complete the search, yielding an UNCERTAIN label. A monolithic formalizer may have access to a strong prover but fail because the original input is globally mistranslated. We therefore use LLM proof trees as *semantic proposal mechanisms* and theorem provers as *local agreement filters*. Rather than trusting an entire generated proof tree or requiring a complete global formalization, our method validates individual candidate derivations and reuses only those accepted by the symbolic stage.

Hybrid Reasoning. We introduce a pipeline that factorises these operations (Figure 1). In **Stage 1**,

an LLM proposes a structured proof tree: a sequence of derivations $d_1, \ldots, d_k$, each citing a support set $S_i \subseteq P \cup \{d_j : j < i\}$, together with a *raw verdict* in $\{\text{TRUE}, \text{FALSE}, \text{UNCERTAIN}\}$. In **Stage 2**, each derivation is checked as its own small entailment problem: its supports become local premises and its claim the local conclusion, both autoformalized and discharged by a prover; we keep only the validated subset $\mathcal{D}^+ = \{d_i : S_i \vdash d_i\}$. In **Stage 3**, an *uncertainty-gated repair* fires *only* when the raw verdict is UNCERTAIN: it attempts to prove h or $\neg h$ from $P \cup \mathcal{D}^+$, relabelling the instance only if one side is discharged, and otherwise leaving it UNCERTAIN. Confident raw TRUE/FALSE verdicts are preserved untouched. The symbolic stage does limited work: it searches and validates within the derivational structure the LLM already proposed.

The autoformalization engine that powers Stage 2 is also a strong standalone baseline, letting us measure exactly what *per-derivation* validation and gated repair add on top of monolithic translation, on identical models and splits.

Contributions.

1. We propose **Hybrid Reasoning**, a three-stage neurosymbolic pipeline that explicitly separates proposing, validating, and deciding a logical NLI label, and validates reasoning *per derivation* rather than per problem (§3).
2. We introduce **uncertainty-gated repair**: a conservative relabelling step that trusts only symbolically validated derivations ($\mathcal{D}^+$) and fires only on the model's own UNCERTAIN verdicts, correcting abstentions without overwriting confident predictions (§3).
3. Across four benchmarks and five LLMs, Hybrid Reasoning is strongest on the hardest datasets (FOLIO and ProverQA), ceiling-level on PrOntoQA, and highly competitive on ProofWriter; direct repair is high-precision and net-positive, complementarity analysis shows $+508$ corrected-minus-regressed predictions, and validation depth predicts reliability monotonically (§5–§6).[1]

2 Related Work

Full-translation neurosymbolic reasoning. Logic-LM (Pan et al., 2023) formulates a problem, hands it to one of several solvers, and self-refines on solver error messages; LINC (Olausson et al., 2023) parses each problem into first-order logic and aggregates k prover calls by majority vote; SatLM (Ye et al., 2023) casts reasoning as satisfiability. All commit to a *single global* formalisation, so one faulty clause can forfeit the whole derivation, and LINC's own analysis attributes a large share of errors to translation faults. Hybrid Reasoning never commits to one global program: it validates and reuses derivations individually, and the global formalisation is retained only as one baseline among several.

Decomposition and compositional translation. CLOVER (Ryu et al., 2025) translates each sentence compositionally and verifies candidate formulas by logical-equivalence or counter-interpretation; Decompose-and-Formalise (Quan et al., 2026) builds an entailment tree, verifies it bottom-up, and locally refines failing steps; EPF (Lee et al., 2025) trains translators to preserve entailment. These verify *translations* or *rewrite* explanations until they pass. Hybrid Reasoning instead asks a different question per step (whether the cited supports actually entail this claim) and *discards* rather than rewrites steps that fail, treating the surviving set as evidence for a gated decision.

Step verification and process supervision. Process reward models (Lightman et al., 2024) and formally-supervised variants (Kamoi et al., 2026) learn to score steps; NLProofS (Yang et al., 2022) searches proofs under a learned verifier. These verifiers are approximate. We replace the learned verifier with a sound prover invoked per derivation, requiring no step-level supervision; the cost is that derivations which fail to autoformalize are simply not promoted to $\mathcal{D}^+$ rather than penalised.

Proof generation and certified decoding. A long line of work generates natural-language proofs or entailment trees (Clark et al., 2021; Dalvi et al., 2021; Saha et al., 2021; Tafjord et al., 2022), and certified decoding constrains generation to be sound (Poesia et al., 2024). Hybrid Reasoning borrows the proof-tree representation but adds a post-hoc symbolic filter, and works with any black-box LLM API. Within the LLM, symbolic chain-of-thought (Xu et al., 2024) and self-consistency (Wang et al., 2023) keep verification internal; we externalise it.

Benchmarks. We evaluate on FOLIO (Han et al., 2024a), ProofWriter (Tafjord et al., 2021),

[1] Code will be released on GitHub upon acceptance.

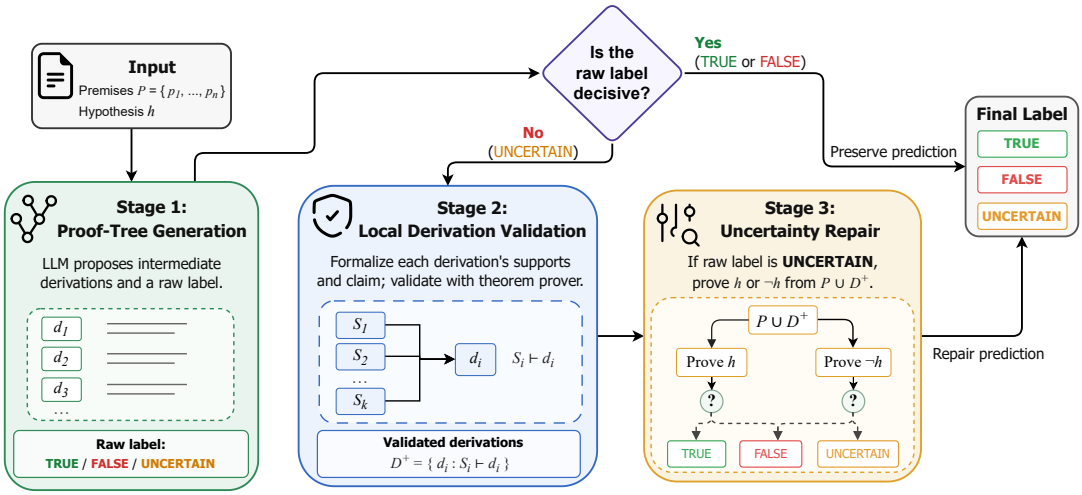

Figure 1: **Hybrid Reasoning. Stage 1** an LLM proposes derivations d_i with supports S_i and a raw verdict. **Stage 2** each derivation is validated as a local entailment problem $S_i \vdash d_i$, yielding the validated set $\mathcal{D}^+ = \{d_i : S_i \vdash d_i\}$. **Stage 3** if the raw verdict is decisive (TRUE/FALSE) the prediction is preserved; otherwise *uncertainty-gated repair* attempts to prove h or $\neg h$ from $P \cup \mathcal{D}^+$, repairing the label only if one side is discharged.

ProverQA (Qi et al., 2025), and PrOntoQA (Saparov and He, 2023); related benchmarks include P-FOLIO (Han et al., 2024b), LogicBench (Parmar et al., 2024), Multi-LogiEval (Patel et al., 2024), and SATBench (Wei et al., 2025). General NLI corpora (Bowman et al., 2015; Williams et al., 2018; Nie et al., 2020) lack the explicit logical structure our symbolic stage requires.

3 Method

3.1 Problem setting and notation

An instance is a triple (P, h, ℓ) where $P = \{p_1, \ldots, p_n\}$ is a set of natural-language premises, h is a hypothesis, and ℓ is the gold label. The label space is $\{\text{TRUE}, \text{FALSE}, \text{UNCERTAIN}\}$ under the open-world reading used by FOLIO, ProofWriter, and ProverQA; PrOntoQA is the binary closed-world case. A method maps (P, h) to a predicted label $\hat{\ell}$.

A derivation is a natural-language claim d_i together with a support set $S_i \subseteq P \cup \{d_j : j < i\}$. We use validation as an operational agreement signal between the proof-tree stage and the symbolic stage. We do not assume that validated derivations constitute faithful explanations of the model's internal reasoning, nor that failed validation always implies semantic invalidity: the prover certifies entailment only relative to the generated formalization.

3.2 Stage 1: Proof-tree generation

The LLM is prompted to emit a structured object rather than free text. It contains: (i) a *goal analysis* naming a `true_goal` and a `false_goal`, each decomposed into subgoals and annotated with a *main connective* describing the conclusion's logical shape (a single literal, a conjunction, or a disjunction); (ii) a *proof tree*, an ordered list of derivations $d_1, \ldots, d_k$ where each d_i records a claim and a support set $S_i \subseteq P \cup \{d_j : j < i\}$ of premise indices and earlier derivations; and (iii) a *raw verdict* $r \in \{\text{TRUE}, \text{FALSE}, \text{UNCERTAIN}\}$. We parse the object, check required fields, and structurally validate that every cited support exists and is acyclic; malformed outputs trigger a single repair-prompt that returns the parser error (full prompt schema and one-shot demonstrations in Appendix I). The raw verdict r is the prediction of an LLM-only "Hybrid Raw" system and is the quantity the later stages may revise.

3.3 Stage 2: Local derivation validation

Each derivation is re-cast as a self-contained entailment problem and checked symbolically. For d_i, its supports S_i become local premises and its claim becomes the local conclusion. The pair is passed to an autoformalization module (the same module used as our standalone *Autoformalization* baseline), which produces a shared vocabulary and first-order formulas, checks vocabulary consistency, lowers the result to the Prover9 theorem prover (McCune,

2005–2010), and runs it. We define the *validated set*

$$\mathcal{D}^+ \;=\; \{\, d_i \;:\; S_i \vdash d_i \,\},$$

i.e. exactly the derivations the prover discharges from their cited supports. Target matches that link derivations to the goal subgoals are validated the same way. A derivation that fails to autoformalize or is not provable is simply *omitted* from $\mathcal{D}^+$; it is never treated as evidence against the model. The per-instance proportion $|\mathcal{D}^+|/k$ yields a *validation status*: *fully*, *partially*, or *not* validated (plus *empty tree*), which we later show predicts reliability.

3.4 Stage 3: Uncertainty-gated repair

The final stage is deliberately conservative. If the raw verdict r is decisive (TRUE or FALSE), Hybrid Reasoning *preserves* it: the validated derivations are not allowed to overturn a confident model prediction. Repair fires only when $r =$ UNCERTAIN. In that case we assemble the repair premise set $P \cup \mathcal{D}^+$ (the original premises augmented with *only* the proved derivations) and attempt to discharge a hierarchy of conclusions on each side: the original conclusion, then the side goal, then its subgoals, combined according to the main connective (a single literal requires its lone subgoal; a disjunctive goal accepts *any* subgoal; a conjunctive goal requires *all*). If the true side is discharged we relabel TRUE; if the false side is discharged, FALSE; if both or neither, the instance remains UNCERTAIN, with a lightweight chain-of-thought fallback as a last resort. Algorithm 1 summarises the procedure.

Algorithm 1 Hybrid Reasoning

Require: premises P, hypothesis h, LLM M, prover $\vdash$
Generate proof tree $T = \{(S_i, d_i)\}_{i=1}^{k}$ and raw label r
$\mathcal{D}^+ \leftarrow \emptyset$
for each derivation (S_i, d_i) in T **do**
  if $S_i \vdash d_i$ **then**
    $\mathcal{D}^+ \leftarrow \mathcal{D}^+ \cup \{d_i\}$
  end if
end for
if $r \in \{\text{TRUE}, \text{FALSE}\}$ **then**
  return r ▷ preserve confident verdict
end if
$P' \leftarrow P \cup \mathcal{D}^+$
if $P' \vdash h$ **then**
  return TRUE
else if $P' \vdash \neg h$ **then**
  return FALSE
else
  return UNCERTAIN
end if

4 Experimental Setup

Datasets. We use balanced subsets of four logical-NLI benchmarks: **FOLIO** (Han et al., 2024a) (182 expert-written, three-label examples), **ProofWriter** (Tafjord et al., 2021) (225, open-world, depth-controlled), **ProverQA** (Qi et al., 2025) (210, the hard split with 6–9 reasoning steps), and **PrOntoQA** (Saparov and He, 2023) (200, binary closed-world). The three open-world sets use the label space {TRUE, FALSE, UNCERTAIN}.

Models. We evaluate five instruction-tuned LLMs spanning vendors and scales: DeepSeek V4 Flash (DeepSeek-AI, 2026), GPT-5 Nano (Singh et al., 2026), Gemini 3.1 Flash-Lite (Google DeepMind, 2026), Grok 4.3 (xAI, 2025), and Qwen 3.5 Flash (Yang et al., 2025). All methods use the same decoding configuration per model; the symbolic stages call Prover9 with a fixed timeout.

Methods. We compare four *internal* methods that we run under identical conditions: **Standard** (label only), **CoT** (short natural-language reasoning then label) (Wei et al., 2022), **Autoformalization** (whole-problem first-order translation discharged by Prover9, the monolithic baseline), and **Hybrid Reasoning**. We additionally report two *external* neurosymbolic baselines, **Logic-LM** (Pan et al., 2023) and **LINC** (Olausson et al., 2023), on the splits where they apply (reproduction details in Appendix D; execution rate, executable accuracy, and final accuracy reported separately in Appendix C). For reference we also report the LLM-only *Hybrid Raw* verdict (the Stage-1 prediction before validation and repair).

Metrics. The primary metric is accuracy. We further report two distinct notions of repair/complementarity. *Direct repair* metrics count only cases where the uncertainty-gated repair stage changes the raw label: repair precision, repair coverage, helpful repairs, harmful repairs, and repair net. *Complementarity* metrics compare Hybrid Final against Hybrid Raw and monolithic Autoformalization over row-level correctness partitions, measuring whether Hybrid preserves raw-only successes, recovers formalization-only successes, or rescues examples both components missed. Per-class precision/recall is reported in Appendix E.

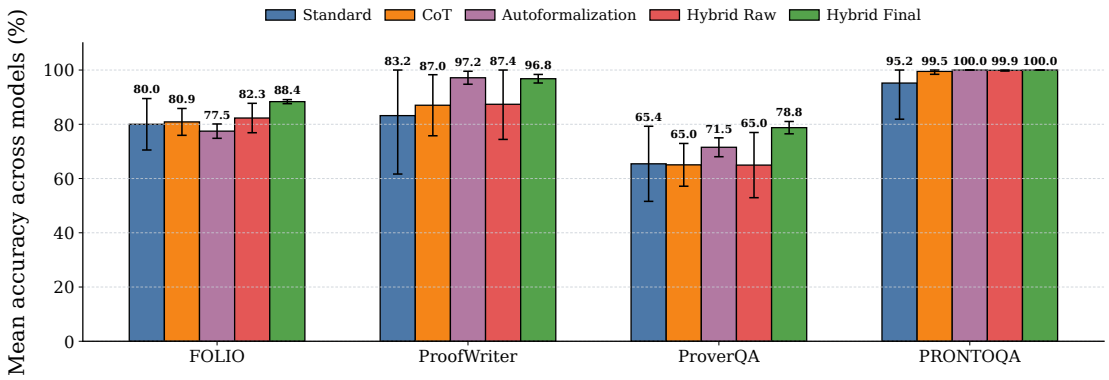

Figure 2: Mean accuracy across the five models for each internal method, with 95% t-confidence intervals. *Hybrid Final* is the validated, repaired pipeline; *Hybrid Raw* is the LLM's Stage-1 verdict before validation and repair. The raw verdict is not reliably better than CoT, but validation and uncertainty-gated repair lift it to the top, especially on ProverQA.

5 Results

Figure 2 summarises internal methods with confidence intervals across models, and Table 1 reports the full accuracy breakdown for every method, model, and dataset.

Hybrid Reasoning leads where the task is hard. On the two hardest, non-saturated benchmarks, Hybrid Reasoning is the best method for *every* model. On FOLIO it improves the mean over the strongest competing method (Logic-LM, 83.1) to 88.4, peaking at 89.0 (Gemini 3.1 Flash-Lite). On ProverQA-hard it raises the mean from 73.9 (Logic-LM) to 78.8 and is best on all five models, peaking at 81.0 (Grok 4.3). These are the regimes where monolithic translation often fails to execute and where CoT is least reliable: long chains, three-way labels, and semantically varied premises.

On easy data it ties the symbolic ceiling. ProofWriter is comparatively easy for whole-problem formalisation, and there the monolithic *Autoformalization* baseline is marginally strongest on average (97.2 vs. Hybrid's 96.8); Hybrid is best or second on every model and never trails the best symbolic method by more than 2.2 points. PrOntoQA is saturated: Hybrid reaches 100% for all models, confirming it does not regress on simple closed-world reasoning even as the weakest baseline (Standard on Gemini, 76.0) collapses.

Where the gains come from. Figure 2 contrasts *Hybrid Raw* (the Stage-1 verdict) with *Hybrid Final*. Raw proof-tree output alone gives only marginal gains over CoT; validation and gated repair drive most of the improvement, most visibly on ProverQA, where the mean rises from a raw 65.0 to a final 78.8. The following analysis explains where this gain comes from, and Appendix H ablates which component (gating, validated derivations, or both) is responsible.

6 Analysis

6.1 Repair is high-precision and net-positive

Table 2 reports direct repair coverage and precision. Repair fires on a minority of instances (from 6.9% on FOLIO to 19.4% on ProverQA) and is correct the large majority of the time it does: precision is 90.5% on FOLIO, 98.2% on ProofWriter, and 78.4% on the much harder ProverQA. Because repair is gated to the UNCERTAIN region, its risk is bounded: across all datasets it produces 326 helpful relabellings against 19 harmful ones, for a direct repair net of +307. The per-dataset net is positive everywhere, translating into accuracy gains of up to +13.8 points on ProverQA. Firing rate scales with task difficulty: the model abstains more often on harder instances, so repair has more opportunities to fire on the cases that benefit from it.

6.2 Validation depth predicts reliability

Figure 3 stratifies instances by validation status, pooled across models (per-model gains in Appendix F.1). The distribution (panel a) shows that most proof trees validate fully or partially. The key result is panel (b): the model's own *raw* accuracy rises monotonically with validation depth: on FOLIO from 78.5% (partially) to 87.4% (fully), on ProofWriter from 65.5% to 91.3%, and on ProverQA from 57.1% to 65.3%. In other words, *how much of a proof tree a prover can indepen-*

Dataset	Model	*External*		*Internal (this work's harness)*			
		Logic-LM	LINC	Standard	CoT	Autoform.	Hybrid
FOLIO	DeepSeek V4 Flash	83.5	81.3	84.6	82.4	79.1	**88.5**
	GPT-5 Nano	78.0	77.5	79.1	76.4	74.7	**87.4**
	Gemini 3.1 Flash-Lite	80.8	74.2	67.0	76.9	78.0	**89.0**
	Grok 4.3	85.7	75.8	84.1	85.2	79.7	**88.5**
	Qwen 3.5 Flash	87.4	81.3	85.2	83.5	75.8	**88.5**
	Mean	*83.1*	*78.0*	*80.0*	*80.9*	*77.5*	***88.4***
ProofWriter	DeepSeek V4 Flash	83.6	96.0	89.8	87.1	95.6	**96.4**
	GPT-5 Nano	78.7	97.3	96.4	96.0	**99.1**	98.7
	Gemini 3.1 Flash-Lite	78.7	**97.3**	53.3	72.9	94.7	95.1
	Grok 4.3	81.3	96.0	83.6	85.3	**98.2**	96.9
	Qwen 3.5 Flash	85.8	95.6	92.9	93.8	**98.2**	96.9
	Mean	*81.6*	*96.4*	*83.2*	*87.0*	***97.2***	*96.8*
ProverQA	DeepSeek V4 Flash	74.8	70.0	71.0	71.4	73.8	**80.0**
	GPT-5 Nano	68.1	63.8	59.5	59.0	69.5	**76.2**
	Gemini 3.1 Flash-Lite	71.9	67.6	49.0	59.0	68.1	**78.1**
	Grok 4.3	79.0	73.8	71.9	63.8	74.8	**81.0**
	Qwen 3.5 Flash	75.7	48.6	76.2	71.9	71.4	**78.6**
	Mean	*73.9*	*64.8*	*65.5*	*65.0*	*71.5*	***78.8***
PrOntoQA	DeepSeek V4 Flash	98.0	**100.0**	**100.0**	99.5	**100.0**	**100.0**
	GPT-5 Nano	93.5	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	Gemini 3.1 Flash-Lite	99.0	**100.0**	76.0	98.0	**100.0**	**100.0**
	Grok 4.3	98.5	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	Qwen 3.5 Flash	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	Mean	*97.8*	***100.0***	*95.2*	*99.5*	***100.0***	***100.0***

Table 1: Accuracy (%) by dataset and model. **Bold** marks the best method in each row, underline the second best. *Autoform.* is whole-problem autoformalization discharged by Prover9 (the monolithic neurosymbolic baseline); *Hybrid* is the full pipeline. PrOntoQA is saturated and serves as a sanity check.

Dataset	Fires	Rate	Prec.	Net	Gain
FOLIO	63	6.9%	90.5%	+55	+6.0
ProofWriter	110	9.8%	98.2%	+106	+9.4
ProverQA	204	19.4%	78.4%	+145	+13.8
PrOntoQA	1	0.1%	100.0%	+1	+0.1

Table 2: Direct uncertainty-gated repair coverage and precision (pooled over models). *Fires*: repair invocations; *Rate*: firing rate; *Prec.*: fraction of firings that reach the gold label; *Net*: helpful minus harmful relabellings among repair firings; *Gain*: raw→final accuracy change (points).

dently discharge is a usable confidence signal for the LLM's verdict, available at inference with no labels. Panel (c) shows the repaired pipeline lifts accuracy at every validation level, with the largest gains exactly where raw reliability is lowest. Monolithic pipelines, which emit a single global program, expose no comparable signal.

6.3 Complementarity with monolithic translation

Does per-derivation validation add anything beyond the monolithic *Autoformalization* baseline? Table 3 answers by partitioning instances on whether the raw verdict and the monolithic formalisation were each correct. When only the monolithic formalisation was right, Hybrid recovers 77.7% of those cases, capturing some of the symbolic-search benefit without depending on a single global program. When only the raw verdict was right, Hybrid preserves it 95.0% of the time. And in 19.1% of cases where both were wrong, Hybrid still reaches the gold label, a gain neither component achieves on its own. This complementarity analysis uses a different accounting from direct repair: its *Net* column is corrected-minus-regressed predictions over row-level partitions, yielding +508 overall.

Figure 4 shows the same mechanism at the label level. The largest movement is out of the UNCERTAIN prediction column and into the correct TRUE/FALSE labels. This matches the gate's intended role: repairing overpredicted uncertainty

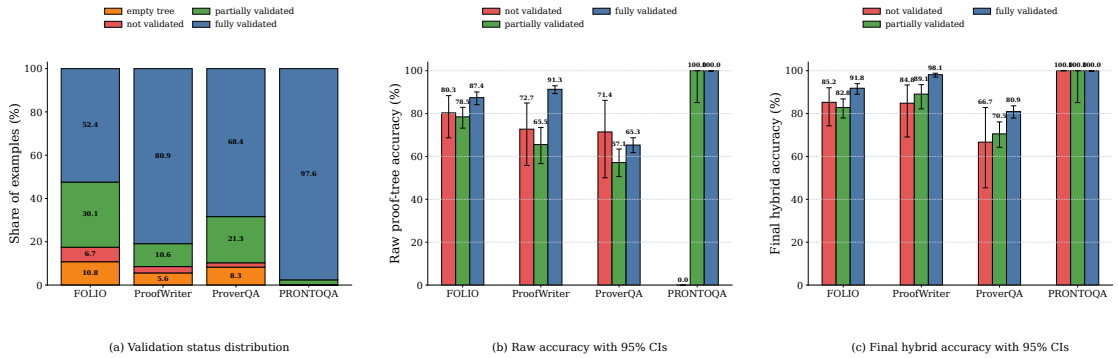

Figure 3: Validation-status analysis, pooled across models. (a) Share of instances by how much of the proof tree validated. (b) The LLM's *raw* accuracy rises monotonically with validation depth, making validation status a label-free reliability signal. (c) The repaired pipeline improves accuracy at every level, most where raw reliability is lowest. Error bars are 95% Wilson intervals.

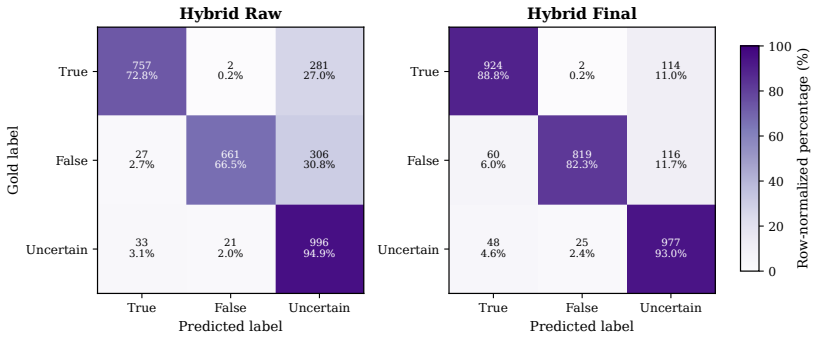

Figure 4: Row-normalised confusion before (Hybrid Raw) and after (Hybrid Final) repair, pooled over the three open-world datasets. Repair reclaims TRUE/FALSE cases the model had left UNCERTAIN (recall 73→89% and 66→82%) while barely touching confident predictions.

Dataset	Form.-only recov.	Raw-only preserv.	Both-wrong rescued	Net
FOLIO	62.5%	100.0%	17.5%	+163
ProofWriter	91.3%	100.0%	11.1%	+111
ProverQA	73.9%	88.6%	20.9%	+233
PrOntoQA	100.0%	–	–	+1
All	**77.7%**	**95.0%**	**19.1%**	**+508**

Table 3: Benefit/risk complementarity. *Form.-only recov.*: of instances only the monolithic formalisation got right, the share Hybrid also gets right. *Raw-only preserv.*: of instances only the raw verdict got right, the share Hybrid preserves. *Both-wrong rescued*: of instances both got wrong, the share Hybrid rescues. *Net*: corrected-minus-regressed predictions over this row-level complementarity accounting, distinct from the direct repair net in Table 2.

rather than relitigating every prediction. Appendix F.2 shows the same effect at the marginal-distribution level.

6.4 Derivation economy

The prover acts as a filter, but most proposed reasoning survives it. Pooled over the open-world datasets, models generate 4.4 derivations per instance, of which 4.1 (93%) validate; the retained fraction ranges from 85% on FOLIO (where commonsense leaps are most frequent) to 95% on ProofWriter and ProverQA. Validation removes roughly 7% of derivations on average, enough to change repair outcomes (Table 3) without discarding most of what the model produced.

6.5 Error modes

The residual errors fall into four types: **(V1)** malformed proof objects (rare on frontier models); **(V2)** vocabulary mismatch between a derivation's supports and its autoformalized form; **(V3)** autoformalization that is syntactically valid but semantically off, so the prover discharges something the model did not intend (related to LINC's L1/L2 error categories, Olausson et al., 2023); and **(V4)** unresolved UNCERTAIN instances where repair fired but

neither side could be proved from $P \cup \mathcal{D}^+$. V2 and V4 dominate on ProverQA and explain its lower repair precision, as reflected in the residual UNCERTAIN-class precision reported in Appendix E.

7 Discussion

Two patterns in the results help explain when Hybrid wins. Proof-tree LLMs are good at identifying relevant local consequences, paraphrases, and intermediate facts, but often cannot stitch them into a complete proof. Whole-problem formalizers can search reliably once the translation is faithful, but a single mistranslated clause anywhere in the input can break execution. Hybrid keeps the LLM's local proposals as a source of evidence and uses the prover only where it has reliable structure to work on. Derivations become semantic proposals; local validation supplies search over the subset that survives.

This framing also explains where the method helps most. The gains concentrate on FOLIO and ProverQA, where naturalistic language, longer chains, and three-way labels create many cases in which the model proposes useful derivations but stops short of a verdict. ProofWriter and PrOntoQA leave little headroom: the former is nearly solved by monolithic Autoformalization, and the latter is saturated by several methods. On the harder benchmarks, uncertainty-gated repair converts local agreement into a determinate label without requiring the prover to trust a full global translation.

The gate is the main lever in the precision-coverage tradeoff. Because Hybrid preserves raw TRUE/FALSE predictions, validated derivations cannot overturn confidently wrong labels. This caps recall on some residual errors, but it also keeps the symbolic stage from acting as an unconstrained second prediction. The combination of high repair precision and a 95% raw-only preservation rate reflects this design choice: the system is built to repair abstentions, not to revisit every prediction. More aggressive repair policies are possible and worth testing, but they make a different risk tradeoff and should be reported separately.

A caveat on what validation means. A fully validated proof tree is not evidence that the LLM internally followed that proof, and a failed validation does not prove the derivation is false. The formalizer can mistranslate a valid step, and multiple valid proof paths may exist for the same problem. The narrower claim is what matters here: when independently formalized derivations are locally provable, they give a label-free reliability signal and a safer basis for repair. This is what monolithic systems cannot offer, since they typically expose only a single executable (or non-executable) global program.

8 Conclusion

We presented Hybrid Reasoning, which separates proposing, validating, and deciding a logical-NLI label, validates reasoning per derivation with a sound prover, and applies an uncertainty-gated repair that trusts only validated derivations and never overwrites confident predictions. The pipeline is strongest where reasoning is hardest, repairs abstentions at high precision and net-positive risk, and exposes a label-free reliability signal through validation depth. The results suggest that checking local agreement between natural-language derivations and symbolic validation is a workable alternative to either unconstrained CoT or whole-problem formalization. Future work includes richer validators (SMT for arithmetic, higher-order provers), more aggressive repair policies with explicit risk controls, and using the validated set $\mathcal{D}^+$ as a source of process-supervision signal for training.

Limitations

Scope of logic. Our symbolic stage targets first-order entailment; temporal, modal, defeasible, and arithmetic reasoning fall outside its scope and would require different solver-translator pairs. The pipeline structure (propose, locally validate, gate) should transfer, but the per-stage components do not.

Autoformalization bottleneck. Validation can fail silently when a derivation is translated into syntactically valid but semantically incorrect first-order logic (error mode V3). Similar failure modes have been observed in prior translation-based systems (Olausson et al., 2023; Pan et al., 2023), and they are intrinsic to any translation-based gate; they bound the precision of our repair stage.

Conservative repair policy. The gate fires only when the raw verdict is UNCERTAIN and refuses to overwrite confident TRUE/FALSE predictions. This asymmetry underwrites our precision numbers but caps recoverable coverage on confidently wrong instances. More aggressive policies, such as confidence-thresholded overwrites, staged arbitration, or learned risk controls, would shift the

precision–coverage frontier and warrant separate evaluation.

Validation depth as a discarded training signal. Reliability rises monotonically with validation depth (§6.2), but we use this signal only at inference time. The validated set $\mathcal{D}^+$ is naturally-labelled step-level data and could supervise process reward models, or be distilled into a single forward pass that approximates the gating behaviour at lower cost.

Saturation. PrOntoQA is at ceiling under strong prompting alone and contributes little discriminative signal; we retain it as a non-regression check rather than a comparative test.

Easy-data ceiling. On ProofWriter the monolithic *Autoformalization* baseline slightly exceeds Hybrid on average. The method's advantage concentrates in harder regimes (FOLIO, ProverQA); we do not claim gains on shallow-depth, fully-formalisable settings.

Inference cost. Cost scales with proof-tree size, since each derivation requires an autoformalization and prover call in Stage 2, and rises further for instances whose raw verdict is UNCERTAIN, which trigger an additional repair pass in Stage 3. Confidently-labelled instances with short trees are relatively cheap; long-chain UNCERTAIN instances incur the full pipeline. Amortising formalisation across derivations, caching prover calls within a problem, or distilling the pipeline into a single model conditioned on validation traces are natural cost-reduction targets.

Language. All experiments are in English; multilingual transfer would depend on both LLM translation quality and the availability of benchmarks with comparable logical structure.

Ethics Statement

This work studies logical reasoning over public benchmarks and involves no human subjects or personal data. The datasets are released for research, and our use is consistent with their intended purpose. Reliance on commercial LLM APIs entails the usual energy and reproducibility considerations noted above.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2021. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7358–7370, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence. Technical Report.

Bruce Frederiksen. 2008. Applying expert system technology to code reuse with Pyke. In *Proceedings of PyCon*, Chicago, IL.

Google DeepMind. 2026. Gemini 3.1 Flash-Lite model card. https://deepmind.google/models/model-cards/gemini-3-1-flash-lite/.

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, and 16 others. 2024a. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031, Miami, Florida, USA. Association for Computational Linguistics.

Simeng Han, Aaron Yu, Rui Shen, Zhenting Qi, Martin Riddell, Wenfei Zhou, Yujie Qiao, Yilun Zhao, Semih Yavuz, Ye Liu, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Dragomir Radev, Rex Ying, and Arman Cohan. 2024b. P-FOLIO: Evaluating and improving logical reasoning with abundant human-written reasoning chains. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16553–16565, Miami, Florida, USA. Association for Computational Linguistics.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Sarkar Snigdha Sarathi Das, Ranran Haoran Zhang, Wenpeng Yin, and Rui Zhang. 2026. Efficient prm training data synthesis via formal verification. In *Findings of the Association for Computational Linguistics: ACL 2026*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on*

Neural Information Processing Systems, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Jinu Lee, Qi Liu, Runzhi Ma, Vincent Han, Ziqi Wang, Heng Ji, and Julia Hockenmaier. 2025. Entailment-preserving first-order logic representations in natural language entailment. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5729–5742, Vienna, Austria. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

W. McCune. 2005–2010. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2020. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4885–4901, Online. Association for Computational Linguistics.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. 2023. 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*, pages 5153–5176, Singapore. Association for Computational Linguistics.

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

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. 2024. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13679–13707, Bangkok, Thailand. Association for Computational Linguistics.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024. Multi-LogiEval: Towards evaluating multi-step logical reasoning ability of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20856–20879, Miami, Florida, USA. Association for Computational Linguistics.

Gabriel Poesia, Kanishk Gandhi, Eric Zelikman, and Noah Goodman. 2024. Certified deductive reasoning with language models. *Transactions on Machine Learning Research*.

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

Xin Quan, Marco Valentino, Louise A. Dennis, and André Freitas. 2026. Decompose-and-formalise: Recursively verifiable natural language inference. *Preprint*, arXiv:2601.19605.

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

Swarnadeep Saha, Prateek Yadav, and Mohit Bansal. 2021. multiPRover: Generating multiple proofs for improved interpretability in rule reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3662–3677, Online. Association for Computational Linguistics.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, Online. Association for Computational Linguistics.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2022. Entailer: Answering questions with faithful and truthful chains of reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2078–2093, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. 2025. SATBench: Benchmarking LLMs' logical reasoning via automated puzzle generation from SAT formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33832–33849, Suzhou, China. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

xAI. 2025. Grok 4 model card. https://data.x.ai/2025-08-20-grok-4-model-card.pdf.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024. Faithful logical reasoning via symbolic chain-of-thought. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13326–13365, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 89–105, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: satisfiability-aided language models using declarative prompting. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

A Dataset details

We evaluate on balanced subsets of four logical-NLI benchmarks. Every method, internal and external, is run on the *identical* split for a given dataset, so all reported numbers are directly comparable; all sampling uses a fixed random seed. Each instance is stored as a natural-language premise set P and a hypothesis h with a gold label, alongside reference first-order-logic annotations where the source provides them. Table 4 summarises the splits.

FOLIO. FOLIO is an expert-written first-order-logic reasoning benchmark whose problems use naturalistic language and a three-way label space $\{\text{TRUE}, \text{FALSE}, \text{UNCERTAIN}\}$. We use the validation split after removing the 22 examples that Olausson et al. (2023) identify as erroneous (unbalanced parentheses, gold-label/FOL mismatches, or premise/FOL count mismatches), yielding the same 182-example set used by LINC. We keep the dataset's original label space; the resulting label distribution is mildly imbalanced (63 TRUE, 54 FALSE, 65 UNCERTAIN). FOLIO has the fewest premises per instance (5.3 on average), but the most linguistic variety.

ProofWriter. ProofWriter is a synthetically generated deductive benchmark with controllable proof depth. We use the open-world-assumption (OWA) portion, which preserves the three-way label space, and draw a depth-stratified, label-balanced subset of 225 examples from the test split: 45 examples at each gold proof depth $d \in \{1, 2, 3, 4, 5\}$ and 75 examples per label. This stratification procedure follows the design used in recent decompose-and-verify work (Quan et al., 2026); because we resample under our own seed, the specific instances differ. ProofWriter instances have many short premises (15.3 on average), which makes whole-problem formalisation comparatively easy.

PrOntoQA. PrOntoQA is a synthetic deductive benchmark with a binary label space. Following the configuration adopted by Logic-LM (Pan et al., 2023)—the hardest *fictional-character*, five-hop subset—we take their released split and draw a label-balanced subset of 200 examples (100 TRUE, 100 FALSE). Because strong prompting already nearly solves this setting (§5), PrOntoQA functions as a saturation and non-regression check rather than a discriminative comparison.

ProverQA. ProverQA pairs LLM-generated natural language with symbolic-prover-validated labels. We use its *hard* split, characterised by long chains (6–9 reasoning steps) and many premises

(18.4 on average), and draw a label-balanced subset of 210 examples from the hard validation set (70 per label). This is the most demanding of the four benchmarks and is where Hybrid Reasoning's gains are largest.

Dataset	N	Labels	#Prem.	Depth
FOLIO	182	3	5.3	—
ProofWriter	225	3	15.3	1–5
PrOntoQA	200	2	17.3	5
ProverQA	210	3	18.4	6–9

Table 4: Evaluation splits. N: examples; *Labels*: size of the label space (3 = TRUE/FALSE/UNCERTAIN, 2 = binary); *#Prem.*: mean premises per instance; *Depth*: gold reasoning depth (where defined). ProofWriter is depth-balanced (45 per depth) and all three-label sets are label-balanced except FOLIO, which keeps its natural distribution.

B Model and decoding details

We used five hosted instruction-tuned backbone models: DeepSeek V4 Flash, Gemini 3.1 Flash-Lite, GPT-5 Nano, Grok 4.3, and Qwen 3.5 Flash. For each backbone, we fixed a single reasoning configuration and used it consistently across *all* methods that call that model: Standard prompting, CoT prompting, Autoformalization, Hybrid Raw, Hybrid Final, and the reproduced external baselines whenever the baseline implementation exposed the same controls. The purpose of this design is to isolate the effect of the reasoning procedure rather than confounding it with model-specific decoding choices or per-method tuning.

Table 5 reports the backbone configurations. DeepSeek V4 Flash was run with reasoning enabled and reasoning effort set to high. Gemini 3.1 Flash-Lite was run with the provider default configuration used by the API, which leaves the model's default thinking behaviour active. GPT-5 Nano was run with reasoning effort set to high. Grok 4.3 was run with reasoning effort set to low. Qwen 3.5 Flash was run with thinking enabled. These settings were chosen before evaluation began and were not changed by dataset, prompt type, or method.

For all internally implemented methods, we did not pass explicit temperature or top-p values. This applies to the label-only baseline, CoT baseline, whole-problem Autoformalization baseline, proof-tree generation stage, and repair-related model calls. We therefore treat sampling as part of the provider default configuration, while keeping the higher-level reasoning or thinking controls fixed as shown in Table 5.

Backbone model	Fixed reasoning configuration
DeepSeek V4 Flash	reasoning enabled; effort high
Gemini 3.1 Flash-Lite	provider default thinking behaviour
GPT-5 Nano	reasoning effort high
Grok 4.3	reasoning effort low
Qwen 3.5 Flash	thinking enabled

Table 5: Backbone model configurations used throughout the evaluation. Except for the LINC reproduction described below, temperature and top-p were left unspecified and provider defaults were used.

The only exception is the LINC reproduction. The original LINC codebase relies on stochastic model sampling to produce multiple formalisations before prover aggregation. To preserve this intended behaviour, we retained LINC's stochastic decoding design and used temperature 0.8 for all LINC runs whose backbone API accepted an external temperature parameter. GPT-5 Nano did not expose an external temperature setting in our configuration; for this backbone, LINC therefore relied on the model's provider-side stochasticity. No additional decoding parameter, reasoning setting, or model-specific prompt variant was tuned for Hybrid Reasoning.

C Executable symbolic reasoning

Figure 5 separates execution rate, executable accuracy, and final accuracy. Execution failures generally correspond to cases where a method cannot obtain a usable structured result, such as a parse failure or a critical formalization error that prevents a prover decision. This distinction is useful for interpreting full-translation baselines: Logic-LM has lower execution rates on FOLIO and ProofWriter, while LINC executes more consistently but remains substantially weaker on ProverQA, showing that executability alone does not guarantee faithful formalization. Hybrid has a different failure profile: execution mainly depends on obtaining a parseable proof-tree output, after which local derivation validation and gated repair determine the final decision.

D External baseline reproduction details

We reproduced Logic-LM (Pan et al., 2023) and LINC (Olausson et al., 2023) using their publicly available codebases. Inputs were serialized from our premise–hypothesis records, and outputs were

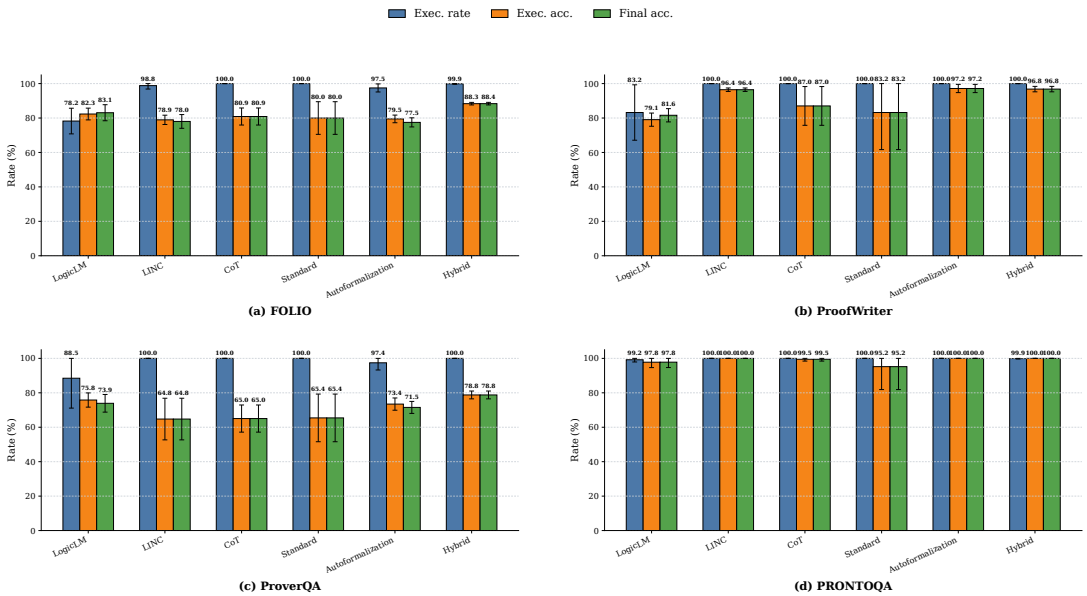

Figure 5: Execution rate, executable accuracy, and final accuracy by dataset and method. Execution rate measures the fraction of instances for which the method obtains an executable symbolic result under its own implementation. Executable accuracy is accuracy restricted to those executable cases; final accuracy includes the method's complete decision procedure, including fallback behavior where applicable.

normalized into the corresponding benchmark label space: TRUE/FALSE/UNCERTAIN for the open-world datasets and binary TRUE/FALSE for PrOntoQA.

Logic-LM. Logic-LM was reproduced following its original solver paths: Prover9 for FOLIO, and Pyke (Frederiksen, 2008) for PrOntoQA and ProofWriter. Since Logic-LM did not include a ProverQA evaluation path, we converted ProverQA examples into the same input format used by its FOLIO pipeline and ran them with the Prover9 solver. We used the strongest setting from the Logic-LM implementation: CoT fallback enabled and self-refinement disabled.

We made a minimal prompt-compatibility patch for modern instruction-tuned models. The original Logic-LM prompts often led recent models to answer the problem directly or add free-form explanatory text, producing outputs that were not parseable by the existing codebase. We therefore kept the few-shot examples, symbolic solver choices, and downstream parser interfaces unchanged, but added explicit final-output constraints to the last query. The following boxes list the exact additions. The patch was intended only to preserve the original completion format expected by Logic-LM's parser.

Logic-LM prompt-compatibility patch.

Logic-LM CoT answer patch: FOLIO

```
For the final example, output only:
1) a short reasoning continuation under "
     Reasoning:"
2) one final line in exactly this format: The
     correct option is: <A/B/C>
Do not add headings, bullet points, markdown, or
     extra answer lines.
```

Logic-LM CoT answer patch: PrOntoQA

```
For the final example, output only:
1) a short reasoning continuation under "
     Reasoning:"
2) one final line in exactly this format: The
     correct option is: <A/B>
Do not add headings, bullet points, markdown, or
     extra answer lines.
```

Logic-LM CoT answer patch: ProofWriter

```
For the final example, output only:
1) a short reasoning continuation under "
     Reasoning:"
2) one final line in exactly this format: The
     correct option is: <A/B/C>
Do not add headings, bullet points, markdown, or
     extra answer lines.
```

Logic-LM FOL formalization patch: FOLIO

```
For the final problem, output only the
     formalization for that problem.
Do not explain your reasoning.
```

```
Do not answer whether the statement is true,
     false, or uncertain.
Do not repeat the previous examples.
Use exactly these three section headers in this
     order and nothing else:
Predicates:
Premises:
Conclusion:
```

Logic-LM rule/fact formalization patch: PrOntoQA

```
For the final problem, output only the
     formalization for that problem.
Do not explain your reasoning.
Do not answer whether the statement is true or
     false.
Do not repeat the previous examples.
Use exactly these four section headers in this
     order and nothing else:
Predicates:
Facts:
Rules:
Query:
```

Logic-LM rule/fact formalization patch: ProofWriter

```
For the final problem, output only the
     formalization for that problem.
Do not explain your reasoning.
Do not answer whether the statement is true,
     false, or uncertain.
Do not repeat the previous examples.
Use exactly these four section headers in this
     order and nothing else:
Predicates:
Facts:
Rules:
Query:
```

LINC. LINC was reproduced without prompt changes or codebase-level changes to its reasoning procedure. We plugged in the evaluated backbone models and kept its original sampling-and-aggregation protocol. The only model-side change was the temperature handling described in Appendix B: we used temperature 0.8 where the model API accepted an external temperature argument, while GPT-5 Nano relied on provider-side stochasticity.

LINC did not include ProverQA or PrOntoQA evaluation paths in the original setup. We therefore converted those examples into the FOLIO-style input format required by its existing pipeline and ran them through the same Prover9-based path.

Execution and final predictions. For both external baselines, execution follows the semantics of the original implementation: an instance is executable when the generated structured formalization can be parsed and passed through the relevant solver path to obtain a usable symbolic result. Final accuracy includes the method's complete decision procedure, including fallback behavior where applicable. This is why Appendix C reports execution rate, executable accuracy, and final accuracy separately.

E Per-label precision and recall

Table 6 reports per-label precision and recall for the monolithic *Autoformalization* baseline, *Hybrid Raw*, and *Hybrid Final*. On the open-world datasets, the main effect of repair is higher TRUE and FALSE recall relative to Hybrid Raw, confirming that the final system primarily recovers determinate cases that the proof-tree stage leaves UNCERTAIN. PrOntoQA is binary, so the UNCERTAIN columns are omitted.

Dataset	Method	TRUE		FALSE		UNCERTAIN	
		P	R	P	R	P	R
FOLIO	Autoform.	90.7	71.4	98.8	61.9	66.6	96.3
	Hybrid Raw	88.3	81.3	94.7	72.2	72.2	91.7
	Hybrid Final	87.9	89.5	95.0	83.7	84.3	91.1
ProofWriter	Autoform.	100.0	95.7	100.0	95.7	92.1	100.0
	Hybrid Raw	99.4	82.1	100.0	80.3	72.8	99.7
	Hybrid Final	98.9	95.5	100.0	95.7	92.1	99.2
ProverQA	Autoform.	82.0	66.6	98.5	55.4	59.8	92.6
	Hybrid Raw	88.9	55.1	93.2	47.1	49.4	92.6
	Hybrid Final	81.4	81.1	94.0	66.9	68.4	88.3
PrOntoQA	Autoform.	100.0	100.0	100.0	100.0	–	–
	Hybrid Raw	100.0	99.8	100.0	100.0	–	–
	Hybrid Final	100.0	100.0	100.0	100.0	–	–

Table 6: Per-label precision/recall (%) for TRUE, FALSE, and UNCERTAIN. PrOntoQA has no UNCERTAIN label.

F Additional analysis

The main text reports pooled repair and complementarity results. Here we include two supplementary diagnostics that are useful for interpreting those results: per-model repair gains and per-dataset prediction distributions.

F.1 Per-model repair gains

Table 7 breaks down the raw-to-final repair gain by model and dataset on the three open-world benchmarks. The gains are not driven by a single backbone: every model improves after uncertainty-gated repair on FOLIO, ProofWriter, and ProverQA. The largest gains occur where the raw proof-tree stage is weakest, most notably Gemini 3.1 Flash-Lite on ProofWriter and ProverQA, and GPT-5 Nano on FOLIO and ProverQA. This pattern is consistent with the intended gating behavior: repair fires where the model has left recoverable cases UNCERTAIN, rather than uniformly rewriting predictions.

Dataset	Model	Raw	Final	Δ	Fires	Helpful	Harmful
FOLIO	DeepSeek V4 Flash	85.2	88.5	+3.3	6	6	0
	GPT-5 Nano	74.7	87.4	+12.6	26	24	1
	Gemini 3.1 Flash-Lite	82.4	89.0	+6.6	14	12	0
	Grok 4.3	84.6	88.5	+3.8	8	7	0
	Qwen 3.5 Flash	84.6	88.5	+3.8	9	8	1
ProofWriter	DeepSeek V4 Flash	94.7	96.4	+1.8	4	4	0
	GPT-5 Nano	96.4	98.7	+2.2	5	5	0
	Gemini 3.1 Flash-Lite	71.1	95.1	+24.0	58	56	2
	Grok 4.3	83.1	96.9	+13.8	31	31	0
	Qwen 3.5 Flash	91.6	96.9	+5.3	12	12	0
ProverQA	DeepSeek V4 Flash	74.3	80.0	+5.7	18	14	2
	GPT-5 Nano	57.6	76.2	+18.6	55	43	4
	Gemini 3.1 Flash-Lite	52.9	78.1	+25.2	66	55	2
	Grok 4.3	65.7	81.0	+15.2	46	36	4
	Qwen 3.5 Flash	74.3	78.6	+4.3	19	12	3

Table 7: Per-model repair breakdown on the open-world datasets. *Raw*/*Final*: accuracy (%) before and after repair; Δ: accuracy gain in points; *Fires*: repair invocations; *Helpful*/*Harmful*: firings that move toward or away from the gold label. PrOntoQA is omitted because it is saturated and has only one repair firing across all models.

F.2 Per-dataset prediction distributions

Figure 6 shows the label-distribution view of the repair mechanism. On the three open-world benchmarks, Hybrid Raw overpredicts UNCERTAIN relative to the gold distribution, most strongly on ProverQA (62.5% predicted UNCERTAIN vs. 33.3% gold). Hybrid Final narrows this gap on every open-world dataset, reducing the predicted UNCERTAIN share to 38.6% on FOLIO, 35.9% on ProofWriter, and 43.0% on ProverQA. The label-distribution shift comes from correcting abstentions, not from reclassifying determinate predictions.

G Implementation details

Structured output and parsing. Hybrid Reasoning uses structured model outputs for the proof-tree stage. Each response is parsed into a goal analysis, an ordered list of derivations, target matches, and a raw verdict. We reject outputs with missing required fields, invalid labels, malformed support references, cyclic dependencies among derivations, or references to derivations that have not yet been introduced. If the final parsed object remains invalid, the instance is marked non-executable for Hybrid and falls back to the CoT fallback prediction.

Retry policy. All reported Hybrid experiments use one retry for proof-tree generation: when the initial structured output fails parsing or structural validation, we pass the parser error back to the model and ask it to return the same object in valid format. Hybrid also uses one retry for each autoformalization subcall: when the initial formalization is unusable, we rerun that subcall once. The standalone *Autoformalization* baseline uses the same one-retry policy for its whole-problem formalization call.

Local validation. For each derivation d_i, the cited support set S_i is converted into a local entailment problem. Premises in S_i may be original input premises or previously generated derivations. The local conclusion is the natural-language claim of d_i. This support–claim pair is passed to the autoformalization module, which generates a first-order vocabulary, formalized premises, and a formalized conclusion. A derivation is added to $\mathcal{D}^+$ only if the resulting formal problem is parseable, vocabulary-consistent, and discharged by Prover9. Failed, non-executable, or unproved local checks are omitted from $\mathcal{D}^+$; they are not treated as negative evidence.

Whole-problem autoformalization. The monolithic *Autoformalization* baseline uses the same formalization and prover interface as local validation, but applies it once to the full premise set and hypothesis. For open-world datasets, the system separately checks whether $P \vdash h$ and whether $P \vdash \neg h$. If exactly one side is proved, the corresponding determinate label is returned; otherwise the prediction is UNCERTAIN. This gives a direct comparison between full-problem translation and Hybrid's local derivation checks under the same prover interface.

Prover calls. All symbolic checks are lowered to Prover9 input and run with a fixed timeout. A

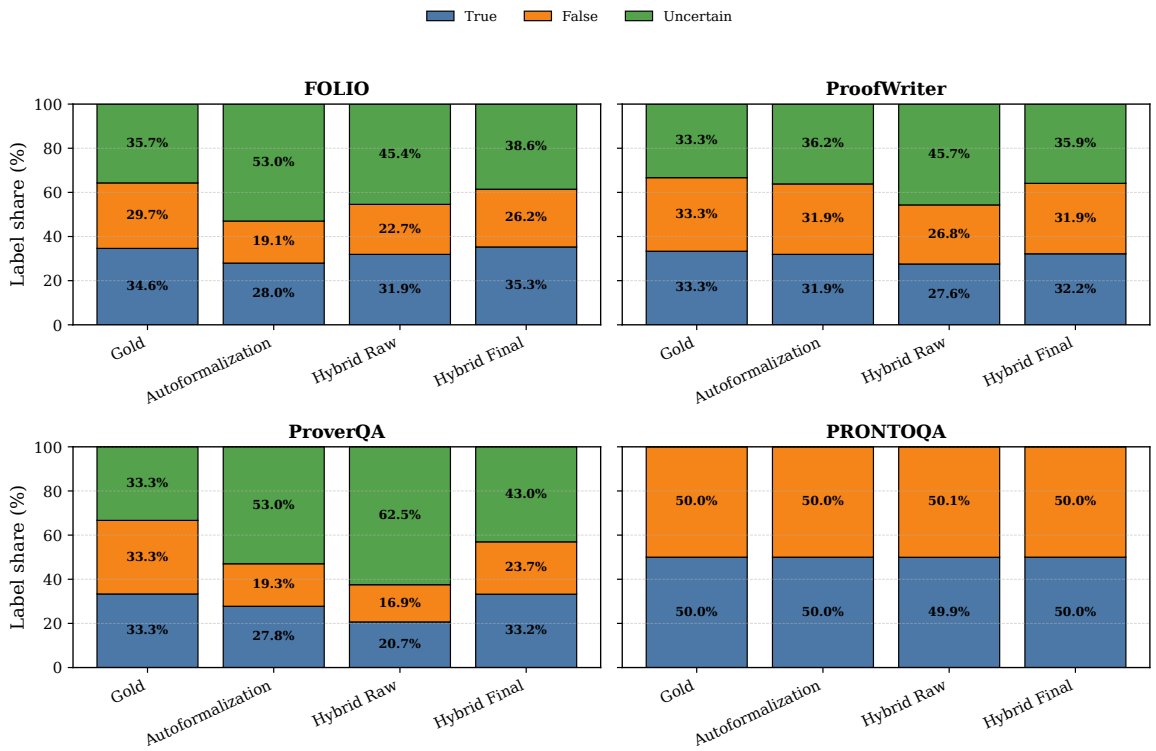

Figure 6: Predicted label distributions for Gold, Autoformalization, Hybrid Raw, and Hybrid Final, per dataset. Hybrid Raw overpredicts UNCERTAIN on the open-world datasets; uncertainty-gated repair shifts mass back toward determinate labels and narrows the gap with the gold distribution.

prover call is counted as successful only when Prover9 (McCune, 2005–2010)returns a proof for the queried entailment. Timeout, syntax failure, vocabulary mismatch, or absence of proof are treated as non-validation for local derivations and as non-execution or non-proof for whole-problem formalization, depending on whether the formal problem was usable by the prover.

Repair decision rule. Repair is applied only when the raw proof-tree verdict is UNCERTAIN. The repair premise set is $P \cup \mathcal{D}^{+}$, i.e., the original premises plus only the locally validated derivations. The system first attempts to prove the original hypothesis side and its negated side; when the proof-tree goal analysis decomposes either side into subgoals, those subgoals are checked according to the recorded connective. A single-literal goal requires its only subgoal, a disjunctive goal accepts any proved subgoal, and a conjunctive goal requires all subgoals. If exactly the true side is discharged, the final label is TRUE; if exactly the false side is discharged, it is FALSE; otherwise it remains UNCERTAIN. Raw TRUE and FALSE predictions are preserved and are not overwritten by repair.

H Ablation study

We ablate the evidence used by the uncertainty-gated repair stage. The main Hybrid system repairs from $P \cup \mathcal{D}^{+}$, where $\mathcal{D}^{+}$ contains only locally validated derivations. We compare it with two variants. The *all-derivations* variant uses every derivation proposed by the proof-tree stage, regardless of whether it was validated. The *no-derivations* variant uses only the original premise set P for repair, so that the repair stage receives no intermediate derivations.

Dataset	Raw	No deriv.	All deriv.	Validated deriv.
FOLIO	82.3	86.9	82.9	**88.4**
ProofWriter	87.4	**97.4**	96.0	96.8
ProverQA	65.0	77.0	75.9	**78.8**
PrOntoQA	99.9	99.9	**100.0**	**100.0**
Mean	83.6	90.3	88.7	**91.0**

Table 8: Repair-evidence ablations, reported as mean accuracy (%) across the five backbone models. *Raw* is the Stage-1 proof-tree verdict before symbolic repair. *No deriv.* repairs using only the original premise set P. *All deriv.* repairs using all proposed derivations, whether validated or not. *Validated deriv.* is the full Hybrid system, which repairs from $P \cup \mathcal{D}^{+}$.

The ablation isolates the role of validated derivations. Using all proposed derivations is not sufficient: it underperforms the validated system on FOLIO and ProverQA, and also trails the no-derivation variant on the open-world datasets. This indicates that intermediate derivations are useful only when they are filtered; otherwise, unvalidated claims can add noise to the repair premise set.

The no-derivation variant is a strong ablation because it tests whether the gain comes merely from rerunning symbolic search on the original problem when the proof-tree model abstains. It improves substantially over the raw verdict, especially on ProofWriter and ProverQA, showing that part of Hybrid's gain comes from giving uncertain instances another symbolic decision opportunity. However, the full validated-derivation system remains best on FOLIO and ProverQA, the two hardest datasets, and is best on average. So Hybrid's gain is doing two things at once: uncertainty gating gives abstaining instances a second symbolic decision opportunity, and the validated derivations add intermediate evidence that the prover would not otherwise have.

I Prompts and few-shot examples

We report the prompt templates and one-shot demonstrations used by the internal methods. Premises are serialized as `P0`, `P1`, ..., and the queried hypothesis is serialized as `C0`. The Standard, CoT, and whole-problem Autoformalization baselines use the shared one-shot demonstration shown below. Hybrid Reasoning uses one dataset-specific one-shot example for each dataset path. For FOLIO, ProofWriter, and ProverQA, the one-shot examples are drawn from publicly available non-evaluation splits of the corresponding datasets. For PrOntoQA, the one-shot example is drawn from the left-out portion of the source split. None of the one-shot examples appears in the corresponding evaluation split.

I.1 Baseline prompts

Standard baseline prompt

```
You solve natural-language logical reasoning
     problems.

You are given:
- A list of premise statements.
- One question statement.

Your task is to decide whether the question is
     entailed by the premises.

Labels:
- "True": the premises entail the question.
- "False": the premises entail the negation of
     the question.
- "Uncertain": the premises entail neither the
     question nor its negation.

Use only the given premises. Do not use outside
     knowledge except for ordinary language
     understanding.

Output valid JSON only using exactly this schema:

{
  "prediction": "True | False | Uncertain"
}
```

Standard baseline one-shot user message

```
Problem:
P0. Lawton Park is a neighbourhood in Seattle.
P1. All citizens of Lawton Park use the zip code
     98199.
P2. Tom is a citizen of Lawton Park.

Question:
C0. Tom uses the zip code 98199.
###
```

Standard baseline one-shot assistant message

```
{
  "prediction": "True"
}
```

CoT baseline prompt

```
You solve natural-language logical reasoning
     problems.

You are given:
- A list of premise statements.
- One question statement.

Your task is to decide whether the question is
     entailed by the premises.

Labels:
- "True": the premises entail the question.
- "False": the premises entail the negation of
     the question.
- "Uncertain": the premises entail neither the
     question nor its negation.

Use only the given premises. Do not use outside
     knowledge except for ordinary language
     understanding.

First write a short reasoning explaining how the
     premises support or fail to support the
     question.
Then write the prediction.

Output valid JSON only using exactly this schema:

{
  "reasoning": "string",
  "prediction": "True | False | Uncertain"
}
```

CoT baseline one-shot user message

```
Problem:
P0. Lawton Park is a neighbourhood in Seattle.
P1. All citizens of Lawton Park use the zip code
     98199.
```

```
P2. Tom is a citizen of Lawton Park.

Question:
C0. Tom uses the zip code 98199.
###
```

CoT baseline one-shot assistant message

```
{
  "reasoning": "P2 states that Tom is a citizen
     of Lawton Park. P1 states that all citizens
     of Lawton Park use the zip code 98199.
     Therefore, Tom uses the zip code 98199.",
  "prediction": "True"
}
```

Whole-problem autoformalization prompt

```
You translate a natural-language reasoning
     problem into first-order logic.

You are given:
- One full problem consisting of premise
     statements.
- One or more conclusion statements to formalize
     against the same problem.

Your task is to output:
1. Predicates/constants used in the formalization
     .
2. Formalized premises.
3. Formalized conclusions.

Use only the predicates and constants you define
     in the output. Do not invent extra symbols
     in formulas.

The grammar of the first-order logic formula is
     defined as follows:

1) logical conjunction of expr1 and expr2: expr1
     ∧ expr2
2) logical disjunction of expr1 and expr2: expr1
     ∨ expr2
3) logical exclusive disjunction of expr1 and
     expr2: expr1 ⊕ expr2
4) logical negation of expr1: ¬expr1
5) expr1 implies expr2: expr1 → expr2
6) expr1 if and only if expr2: expr1 ↔ expr2
7) logical universal quantification: ∀x
8) logical existential quantification: ∃x

Symbol rules:
- Predicate names must use PascalCase.
- Constants must use lowercase or snake_case.
- Every predicate used in a formula must appear
     in "predicates".
- Every constant used in a formula must appear in
      "constants".
- Predicate arity must match its use in formulas.
- Use the same symbol consistently across
     premises and conclusions.
- Formalize every premise statement and every
     conclusion statement exactly once.
- Premise IDs must match the numbered premise
     statements you are given: "P0", "P1", ..., "
     Pn".
- Conclusion IDs must match the numbered
     conclusion statements you are given: "C0", "
     C1", ..., "Ck".
- Do not invent extra output fields.
- Do not include natural-language explanations
     outside JSON.
- Do not add explanations, commentary, or
     markdown.

Output exactly one JSON object with this schema:

{
  "predicates": [
    {
      "name": "PredicateName",
      "arity": 1,
      "description": "x is ..."
    }
  ],
  "constants": [
    {
      "name": "constant_name",
      "description": "..."
    }
  ],
  "premises": [
    {
      "id": "P0",
      "formula": "..."
    }
  ],
  "conclusions": [
    {
      "id": "C0",
      "formula": "..."
    }
  ]
}

Output JSON only.
```

Whole-problem autoformalization one-shot user message

```
Problem:
P0. Lawton Park is a neighbourhood in Seattle.
P1. All citizens of Lawton Park use the zip code
     98199.
P2. Tom is a citizen of Lawton Park.

Question:
C0. Tom uses the zip code 98199.
C1. Daniel uses the zip code 98199.
###
```

Whole-problem autoformalization one-shot assistant message

```
{
  "predicates": [
    {
      "name": "NeighbourhoodIn",
      "arity": 2,
      "description": "x is a neighbourhood in y"
    },
    {
      "name": "CitizenOf",
      "arity": 2,
      "description": "x is a citizen of y"
    },
    {
      "name": "UsesZipCode",
      "arity": 2,
      "description": "x uses zip code y"
    }
  ],
  "constants": [
    {
      "name": "lawton_park",
      "description": "Lawton Park"
    },
    {
      "name": "seattle",
      "description": "Seattle"
    },
    {
      "name": "tom",
      "description": "Tom"
    },
    {
      "name": "daniel",
      "description": "Daniel"
    },
    {
```

```
      "name": "zip_98199",
      "description": "the zip code 98199"
    }
  ],
  "premises": [
    {
      "id": "P0",
      "formula": "NeighbourhoodIn(lawton_park,
     seattle)"
    },
    {
      "id": "P1",
      "formula": "∀x (CitizenOf(x, lawton_park)
     → UsesZipCode(x, zip_98199))"
    },
    {
      "id": "P2",
      "formula": "CitizenOf(tom, lawton_park)"
    }
  ],
  "conclusions": [
    {
      "id": "C0",
      "formula": "UsesZipCode(tom, zip_98199)"
    },
    {
      "id": "C1",
      "formula": "UsesZipCode(daniel, zip_98199)"
    }
  ]
}
```

I.2 Hybrid prompts

The Hybrid proof-tree prompt is used for Stage 1. Local derivation validation and target-side repair use the same autoformalization interface as the whole-problem Autoformalization baseline, but with the local supports as premises and the derivation claim or target statement as the conclusion.

Hybrid proof-tree prompt

```
You solve a natural-language logical reasoning
     problem.

You are given:
- A list of premise statements.
- One conclusion statement.

Your task is to produce a compact proof tree and
     predict whether the conclusion is entailed
     by the premises.

Labels:
- "True": the premises entail the conclusion.
- "False": the premises entail the negation of
     the conclusion.
- "Uncertain": neither the conclusion nor its
     negation is entailed.

Goal rules:
- For an atomic conclusion A:
  True goal is A.
  False goal is not A.
- For a negated conclusion not A:
  True goal is not A.
  False goal is A.
- For a conjunction A and B:
  True goal requires both A and B.
  False goal requires not A or not B.
- For a disjunction A or B:
  True goal requires A or B.
  False goal requires both not A and not B.
- For an implication if A then B:
  Treat the true goal as not A or B.
  Treat the false goal as A and not B.
- For a biconditional A iff B:
  True goal requires both directions.
  False goal requires one side true and the other
     side false.
- For a universal conclusion:
  True goal is the universal statement.
  False goal is a counterexample.
- For an existential conclusion:
  True goal is a witness or the existential
     statement.
  False goal is that no such object exists.

The field composition controls how subgoals are
     interpreted:
- "single": prove the statement itself or the
     single main subgoal.
- "any": proving any subgoal satisfies the goal.
- "all": every subgoal must be proved to satisfy
     the goal.

Important:
- Output JSON only.
- Do not output formal logic.
- Derivation claims must be natural-language
     statements, not formulas.
- Even if you cannot fully prove the conclusion,
     generate useful intermediate derivations
     from the premises whenever possible.

Output exactly one JSON object with this schema:

{
  "goal_analysis": {
    "main_connective": "atomic | negation |
     conjunction | disjunction |
     implication_as_disjunction | biconditional |
      universal | existential | complex",
    "true_goal": {
      "reasoning": "string",
      "composition": "single | any | all",
      "statement": "string",
      "subgoals": [
        {
          "reasoning": "string",
          "id": "T1",
          "role": "string",
          "statement": "string"
        }
      ]
    },
    "false_goal": {
      "reasoning": "string",
      "composition": "single | any | all",
      "statement": "string",
      "subgoals": [
        {
          "reasoning": "string",
          "id": "F1",
          "role": "string",
          "statement": "string"
        }
      ]
    }
  },
  "proof_tree": {
    "derivations": [
      {
        "reasoning": "string",
        "id": "D1",
        "claim": "string",
        "support": ["P0", "D1"],
        "rule": "direct_premise |
     universal_instantiation |
     existential_instantiation |
     existential_introduction | modus_ponens |
     modus_tollens | conjunction_elimination |
     conjunction_introduction |
     disjunction_introduction |
     disjunctive_syllogism |
     hypothetical_syllogism |
     biconditional_elimination | contradiction |
     counterexample | semantic_equivalence |
     compound"
      }
    ],
    "target_matches": [
      {
        "reasoning": "string",
```

```
        "target": "T | F | T1 | F1",
        "derivations": ["D1"],
        "match_type": "exact | equivalent |
     entails"
      }
    ],
    "verdict": {
      "reasoning": "string",
      "label": "True | False | Uncertain",
      "satisfied_target": "T | F | none"
    }
  }
}

Rules:
- Use [] for derivations if no useful
     intermediate statement can be derived.
- Prefer generating useful intermediate
     derivations over leaving the proof tree
     empty.
- Use [] for target_matches if no derivation
     clearly matches a proof goal.
- A derivation may only support later derivations
     , never earlier ones.
- support entries must be IDs such as P0, P1, D1,
      D2.
- target_matches should point to derivations that
      satisfy the target or subgoal.
- target_matches.target must be one of T, F, T1,
     T2, ..., F1, F2, ...
- Every subgoal statement must be a standalone
     proposition, not an explanation.
- Put reasoning first inside each object and put
     explanations there, not inside statement
     fields.
```

I.3 FOLIO Hybrid one-shot demonstration

FOLIO Hybrid one-shot user message

```
Problem:
P0. An animal is either a cat or a dog.
P1. All dogs bark.
P2. Ghosts do not exist.
P3. If something barks, then it is not alive.
P4. A thing is either alive or dead.
P5. Jojo is not dead.

Question:
C0. If Jojo is a cat or a ghost, then Jojo either
      barks or is a dog.
###
```

FOLIO Hybrid one-shot assistant message

```
{
  "goal_analysis": {
    "main_connective": "
     implication_as_disjunction",
    "true_goal": {
      "reasoning": "The conclusion is an
     implication A -> B. In this schema, the true
      goal is represented as not A or B.",
      "composition": "any",
      "statement": "Jojo is not a cat and Jojo is
      not a ghost, or Jojo either barks or is a
     dog.",
      "subgoals": [
        {
          "reasoning": "This proves the
     implication true by making the antecedent
     false.",
          "id": "T1",
          "role": "negated_antecedent",
          "statement": "Jojo is not a cat and
     Jojo is not a ghost."
        },
        {
          "reasoning": "This proves the
     implication true by deriving the consequent
     .",
          "id": "T2",
          "role": "consequent",
          "statement": "Jojo either barks or is a
     dog."
        }
      ]
    },
    "false_goal": {
      "reasoning": "An implication A -> B is
     false when A is true and B is false.",
      "composition": "all",
      "statement": "Jojo is a cat or a ghost, and
     Jojo neither barks nor is a dog.",
      "subgoals": [
        {
          "reasoning": "This makes the antecedent
     true.",
          "id": "F1",
          "role": "antecedent",
          "statement": "Jojo is a cat or a ghost
     ."
        },
        {
          "reasoning": "This makes the consequent
     false.",
          "id": "F2",
          "role": "negated_consequent",
          "statement": "Jojo does not bark and
     Jojo is not a dog."
        }
      ]
    }
  },
  "proof_tree": {
    "derivations": [
      {
        "reasoning": "P5 says Jojo is not dead.
     P4 says a thing is either alive or dead, so
     Jojo is alive.",
        "id": "D1",
        "claim": "Jojo is alive.",
        "support": ["P4", "P5"],
        "rule": "disjunctive_syllogism"
      },
      {
        "reasoning": "P3 says that if something
     barks, then it is not alive. Since Jojo is
     alive, Jojo does not bark.",
        "id": "D2",
        "claim": "Jojo does not bark.",
        "support": ["P3", "D1"],
        "rule": "modus_tollens"
      },
      {
        "reasoning": "P1 says all dogs bark.
     Since Jojo does not bark, Jojo is not a dog
     .",
        "id": "D3",
        "claim": "Jojo is not a dog.",
        "support": ["P1", "D2"],
        "rule": "modus_tollens"
      },
      {
        "reasoning": "P0 says an animal is either
      a cat or a dog. Since Jojo is not a dog,
     Jojo is a cat.",
        "id": "D4",
        "claim": "Jojo is a cat.",
        "support": ["P0", "D3"],
        "rule": "disjunctive_syllogism"
      },
      {
        "reasoning": "If Jojo is a cat, then Jojo
      is a cat or a ghost.",
        "id": "D5",
        "claim": "Jojo is a cat or a ghost.",
        "support": ["D4"],
        "rule": "disjunction_introduction"
      },
      {
        "reasoning": "Combining Jojo does not
     bark and Jojo is not a dog gives the
     negation of the consequent.",
        "id": "D6",
        "claim": "Jojo does not bark and Jojo is
     not a dog.",
```

```
        "support": ["D2", "D3"],
        "rule": "conjunction_introduction"
      }
    ],
    "target_matches": [
      {
        "reasoning": "D5 satisfies the antecedent
      subgoal.",
        "target": "F1",
        "derivations": ["D5"],
        "match_type": "exact"
      },
      {
        "reasoning": "D6 satisfies the negated
     consequent subgoal.",
        "target": "F2",
        "derivations": ["D6"],
        "match_type": "exact"
      },
      {
        "reasoning": "F1 and F2 together satisfy
     the false goal.",
        "target": "F",
        "derivations": ["D5", "D6"],
        "match_type": "entails"
      }
    ],
    "verdict": {
      "reasoning": "The antecedent is derivable
     and the consequent is refuted, so the
     implication is false.",
      "label": "False",
      "satisfied_target": "F"
    }
  }
}
```


I.4 ProofWriter Hybrid one-shot demonstration

ProofWriter Hybrid one-shot user message

```
Problem:
P0. Erin is smart.
P1. If something is furry then it is not blue.
P2. Nice things are not cold.
P3. Furry things are cold.
P4. White things are cold.
P5. If something is cold then it is not white.
P6. Smart things are furry.
P7. If something is blue then it is quiet.

Question:
C0. Erin is not white.
###
```

ProofWriter Hybrid one-shot assistant message


```
{
  "goal_analysis": {
    "main_connective": "negation",
    "true_goal": {
      "reasoning": "The conclusion is a negated
     atomic statement, so the true goal is to
     derive that Erin is not white.",
      "composition": "single",
      "statement": "Erin is not white.",
      "subgoals": [
        {
          "reasoning": "This is the negated
     atomic conclusion itself.",
          "id": "T1",
          "role": "main_goal",
          "statement": "Erin is not white."
        }
      ]
    },
    "false_goal": {
      "reasoning": "To make a negated atomic
     conclusion false, the corresponding positive
      atomic statement must be true.",
      "composition": "single",
      "statement": "Erin is white.",
      "subgoals": [
        {
          "reasoning": "This is the positive form
      of the conclusion.",
          "id": "F1",
          "role": "positive_goal",
          "statement": "Erin is white."
        }
      ]
    }
  },
  "proof_tree": {
    "derivations": [
      {
        "reasoning": "P0 directly states that
     Erin is smart.",
        "id": "D1",
        "claim": "Erin is smart.",
        "support": ["P0"],
        "rule": "direct_premise"
      },
      {
        "reasoning": "P6 says smart things are
     furry. Since Erin is smart, Erin is furry.",
        "id": "D2",
        "claim": "Erin is furry.",
        "support": ["P6", "D1"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P3 says furry things are
     cold. Since Erin is furry, Erin is cold.",
        "id": "D3",
        "claim": "Erin is cold.",
        "support": ["P3", "D2"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P5 says that if something
     is cold then it is not white. Since Erin is
     cold, Erin is not white.",
        "id": "D4",
        "claim": "Erin is not white.",
        "support": ["P5", "D3"],
        "rule": "modus_ponens"
      }
    ],
    "target_matches": [
      {
        "reasoning": "D4 directly matches the
     true goal subgoal.",
        "target": "T1",
        "derivations": ["D4"],
        "match_type": "exact"
      },
      {
        "reasoning": "D4 directly proves the true
      goal.",
        "target": "T",
        "derivations": ["D4"],
        "match_type": "exact"
      }
    ],
    "verdict": {
      "reasoning": "The derivation chain proves
     Erin is not white, so the negated atomic
     conclusion is true.",
      "label": "True",
      "satisfied_target": "T"
    }
  }
}
```

I.5 PrOntoQA Hybrid one-shot demonstration

PrOntoQA Hybrid one-shot user message

```
Problem:
P0. Jompuses are not shy.
P1. Jompuses are yumpuses.
P2. Each yumpus is aggressive.
P3. Each yumpus is a dumpus.
P4. Dumpuses are not wooden.
P5. Dumpuses are wumpuses.
P6. Wumpuses are red.
P7. Every wumpus is an impus.
P8. Each impus is opaque.
P9. Impuses are tumpuses.
P10. Numpuses are sour.
P11. Tumpuses are not sour.
P12. Tumpuses are vumpuses.
P13. Vumpuses are earthy.
P14. Every vumpus is a zumpus.
P15. Zumpuses are small.
P16. Zumpuses are rompuses.
P17. Max is a yumpus.

Question:
C0. Max is sour.
###
```

PrOntoQA Hybrid one-shot assistant message


```
{
  "goal_analysis": {
    "main_connective": "atomic",
    "true_goal": {
      "reasoning": "The conclusion is an atomic
     statement, so the true goal is to derive
     that Max is sour.",
      "composition": "single",
      "statement": "Max is sour.",
      "subgoals": [
        {
          "reasoning": "This is the atomic
     conclusion itself.",
          "id": "T1",
          "role": "main_goal",
          "statement": "Max is sour."
        }
      ]
    },
    "false_goal": {
      "reasoning": "For an atomic conclusion, the
      false goal is the negated atomic statement
     .",
      "composition": "single",
      "statement": "Max is not sour.",
      "subgoals": [
        {
          "reasoning": "This is the negation of
     the atomic conclusion.",
          "id": "F1",
          "role": "negated_goal",
          "statement": "Max is not sour."
        }
      ]
    }
  },
  "proof_tree": {
    "derivations": [
      {
        "reasoning": "P17 directly states that
     Max is a yumpus.",
        "id": "D1",
        "claim": "Max is a yumpus.",
        "support": ["P17"],
        "rule": "direct_premise"
      },
      {
        "reasoning": "P3 says each yumpus is a
     dumpus. Since Max is a yumpus, Max is a
     dumpus.",
        "id": "D2",
        "claim": "Max is a dumpus.",
        "support": ["P3", "D1"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P5 says dumpuses are
     wumpuses. Since Max is a dumpus, Max is a
     wumpus.",
        "id": "D3",
        "claim": "Max is a wumpus.",
        "support": ["P5", "D2"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P7 says every wumpus is an
     impus. Since Max is a wumpus, Max is an
     impus.",
        "id": "D4",
        "claim": "Max is an impus.",
        "support": ["P7", "D3"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P9 says impuses are
     tumpuses. Since Max is an impus, Max is a
     tumpus.",
        "id": "D5",
        "claim": "Max is a tumpus.",
        "support": ["P9", "D4"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P11 says tumpuses are not
     sour. Since Max is a tumpus, Max is not sour
     .",
        "id": "D6",
        "claim": "Max is not sour.",
        "support": ["P11", "D5"],
        "rule": "modus_ponens"
      }
    ],
    "target_matches": [
      {
        "reasoning": "D6 directly matches the
     false-goal subgoal.",
        "target": "F1",
        "derivations": ["D6"],
        "match_type": "exact"
      },
      {
        "reasoning": "D6 directly proves the
     false goal.",
        "target": "F",
        "derivations": ["D6"],
        "match_type": "exact"
      }
    ],
    "verdict": {
      "reasoning": "The premises derive the
     negation of the conclusion, so the
     conclusion is false.",
      "label": "False",
      "satisfied_target": "F"
    }
  }
}
```


I.6 ProverQA Hybrid one-shot demonstration

ProverQA Hybrid one-shot user message

```
Problem:
P0. Reece earns recognition.
P1. If Simon stays updated on his field or
     networks with peers, then he will attend
     conferences.
P2. Simon reads medical literature.
P3. Rhys teaches medical students.
P4. For Rhys, advancing the medical field and
     teaching medical students are mutually
     exclusive pursuits.
P5. If Simon has expertise, then he will publish
     papers.
```

```
P6. If Simon has curiosity, then he will pursue
     medical knowledge.
P7. Simon earns recognition.
P8. If someone earns recognition in their field,
     then they have expertise.
P9. Milan reads medical literature.
P10. If Milan publishes papers or collaborates
     with peers, then he attends conferences.
P11. If Milan has expertise, then he will publish
      papers.
P12. If Reece dedicates himself to research, then
      he can either make groundbreaking
     discoveries or improve patient care, but not
      both.
P13. If Simon publishes papers or collaborates
     with peers, then he attends conferences.
P14. Anyone who reads medical literature or
     attends conferences pursues medical
     knowledge.
P15. Anyone who conducts experiments and collects
      data can develop expertise.

Question:
C0. Simon does not attend conferences.
###
```

ProverQA Hybrid one-shot assistant message


```
{
  "goal_analysis": {
    "main_connective": "negation",
    "true_goal": {
      "reasoning": "The conclusion is a negated
     atomic statement, so the true goal is to
     derive that Simon does not attend
     conferences.",
      "composition": "single",
      "statement": "Simon does not attend
     conferences.",
      "subgoals": [
        {
          "reasoning": "This is the negated
     atomic conclusion itself.",
          "id": "T1",
          "role": "main_goal",
          "statement": "Simon does not attend
     conferences."
        }
      ]
    },
    "false_goal": {
      "reasoning": "To make a negated atomic
     conclusion false, the corresponding positive
      atomic statement must be true.",
      "composition": "single",
      "statement": "Simon attends conferences.",
      "subgoals": [
        {
          "reasoning": "This is the positive form
      of the conclusion.",
          "id": "F1",
          "role": "positive_goal",
          "statement": "Simon attends conferences
     ."
        }
      ]
    }
  },
  "proof_tree": {
    "derivations": [
      {
        "reasoning": "P7 directly states that
     Simon earns recognition.",
        "id": "D1",
        "claim": "Simon earns recognition.",
        "support": ["P7"],
        "rule": "direct_premise"
      },
      {
        "reasoning": "P8 says that if someone
     earns recognition in their field, then they
     have expertise. Since Simon earns
     recognition, Simon has expertise.",
        "id": "D2",
        "claim": "Simon has expertise.",
        "support": ["P8", "D1"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "P5 says that if Simon has
     expertise, then he will publish papers.
     Since Simon has expertise, Simon publishes
     papers.",
        "id": "D3",
        "claim": "Simon publishes papers.",
        "support": ["P5", "D2"],
        "rule": "modus_ponens"
      },
      {
        "reasoning": "If Simon publishes papers,
     then Simon either publishes papers or
     collaborates with peers.",
        "id": "D4",
        "claim": "Simon publishes papers or
     collaborates with peers.",
        "support": ["D3"],
        "rule": "disjunction_introduction"
      },
      {
        "reasoning": "P13 says that if Simon
     publishes papers or collaborates with peers,
      then he attends conferences. Since Simon
     publishes papers or collaborates with peers,
      Simon attends conferences.",
        "id": "D5",
        "claim": "Simon attends conferences.",
        "support": ["P13", "D4"],
        "rule": "modus_ponens"
      }
    ],
    "target_matches": [
      {
        "reasoning": "D5 directly matches the
     false-goal subgoal.",
        "target": "F1",
        "derivations": ["D5"],
        "match_type": "exact"
      },
      {
        "reasoning": "D5 directly proves the
     false goal.",
        "target": "F",
        "derivations": ["D5"],
        "match_type": "exact"
      }
    ],
    "verdict": {
      "reasoning": "The premises derive that
     Simon attends conferences, which is the
     negation of the conclusion. Therefore the
     conclusion is false.",
      "label": "False",
      "satisfied_target": "F"
    }
  }
}
```


I.7 Retry prompt

The same retry-message template is used when a structured response cannot be parsed or otherwise cannot be used by the pipeline; the previous output and the parser or validation error are substituted into the two placeholders.

Retry prompt

```
Your previous response could not be used.

Previous output:
{previous_output}

Error:
```

```
{error_message}

Return corrected JSON only.
```