AUTOFORMALIZATION VIA A LAYERED PROTOCOL

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoformalization translates informal language into formal representations whose correctness is machine-verifiable through logical solvers (e.g., SMT solvers, theorem provers, temporal reasoners). This capability is increasingly important as LLMs generate code, executable programs, and actionable decisions that must faithfully satisfy their intended formally representable specifications. While promising, existing methods are largely one-shot, entangling distinct error modalities into a single undifferentiated failure that obscures both where systems fail and how to address them. We introduce a hierarchical taxonomy of autoformalization errors, organized along three modes : syntax, semantics, and grounding. Building on this taxonomy, we propose a layered approach that detects and resolves errors hierarchically, addressing each error class at the stage where it arises rather than at the end of the pipeline. We show that this approach significantly outperforms state-of-the-art methods across three benchmarks spanning first-order logic, theorem proving, and time-series applications.

1 INTRODUCTION

Autoformalization translates informal language to a formal syntax consistent with underlying semantics (Mensfelt et al., 2026). Unlike natural-language outputs, formal representations can be mechanically evaluated using symbolic reasoning engines such as SMT solvers (Pan et al., 2023), theorem provers (Yu et al., 2026), and temporal reasoning engines (Liu et al., 2023a; English et al., 2025). This provides a bridge between the flexibility of natural-language (NL) interaction and the guarantee of formal reasoning: specifications can be expressed in NL while their formal counterparts can be executed, checked for consistency, or verified against explicit rules. This capability is increasingly important as large language models (LLMs) are used to generate code, executable programs, and actionable decisions that must faithfully satisfy intended, formally representable specifications Liu et al. (2023b); Ma et al. (2025); Ye et al. (2026).

Despite recent progress, existing autoformalization methods are largely one-shot: they generate a formal representation and evaluate its correctness only at the end of the process. As a result, fundamentally different error modes are collapsed into a single notion of failure, obscuring both where errors arise and how they should be corrected. We observe that autoformalization failures occur at distinct levels of the translation process and therefore should not be treated uniformly. In particular, errors can arise from malformed formal expressions, incorrect interpretations of the intended meaning, or failures to connect that meaning to the appropriate elements of the desired formalism. An autoformalizer must form a valid expression in the language, preserve the meaning of the informal input, and correctly associate input concepts with the symbols of the formal representation. Based on this observation, we make three contributions. First, we introduce a hierarchical taxonomy that characterizes these failures as syntax, semantics, and grounding errors. Second, we introduce a layered autoformalization approach that detects and resolves errors hierarchically. Rather than correcting an undifferentiated failure at the end of the pipeline, each layer operates on outputs that have satisfied the requirements of preceding layers, enabling errors to be localized and corrected before they propagate downstream. Third, we evaluate our approach across three benchmarks spanning first-order logical reasoning, theorem proving, and temporal use-case, using five LLMs and multiple symbolic reasoning engines. Across these settings, our layered approach reduces average formalization errors by 93.0% on FOLIO, 91.6% on ProofWriter, and 47.6% on τ^3-bench relative to one-shot generation. Beyond aggregate improvements, our layer-wise analysis reveals that the dominant source of failure varies across tasks: errors concentrate in reasoning on FOLIO, remain low

across grounding and reasoning on ProofWriter, and shift toward retrieval and state maintenance in the interactive τ^3-bench setting. These results support our central hypothesis that autoformalization failures arise at distinct layers and benefit from detection and correction at the corresponding layer.

2 Background and Related Work

Autoformalization is the automated translation of informal input into a formal language. A formal language consists of a well-defined alphabet of symbols, syntax rules for constructing valid expressions, and a semantics that assigns meaning to those expressions. For example, in first-order logic, the alphabet includes variables, constants, function symbols, predicate symbols, connectives, quantifiers, and punctuation symbols. We also use domain types over subset of of constants. Using these symbols, terms and well-formed formulas are built inductively according to precise formation rules, and the language is defined as the set of all such formulas. Semantics is provided through interpretations that assign meanings to constants, functions, and predicates over a domain, enabling formulas to be evaluated based on a truth values (e.g., Boolean, probabilistic).

Early approaches to autoformalization relied on rule-based systems and neural machine translation, often requiring controlled language or informal–formal corpora. Recently, LLMs have enabled the translation of NL specifications into formal representations from only a small number of demonstrations (Wu et al., 2022; Yang et al., 2025; 2024; Gao et al., 2025). Synthetic data generation, back-translation, and interaction with formal environments have been used to improve the fidelity of generated formalizations (Yang et al., 2025; Sileo, 2024; Wu et al., 2026). While prior work has focused on formalizing mathematics in proof assistants, recent work extends to translating informal language into logical representations for reasoning, planning, verification, and knowledge representation (Mensfelt et al., 2026). Our work adopts this view, while focusing on a complementary problem: characterizing failures that cascade inherent processes in the formalization. Recent approaches differ in how they construct and refine formal representations. These range from zero shot LLM-based translation (Zheng et al., 2026) to structured generation that use abstract syntax trees before producing logic (Putra et al., 2026), as well as feedback-guided refinement of generated formalizations (Li et al., 2026). Other methods introduce feedback into the formalization process: LLM critics can identify rule violations in conversations (Kang et al., 2026), while symbolic solvers provide execution error messages (Pan et al., 2023; Putra et al., 2026; Pei et al., 2025). FormalRx (Wang et al., 2026) further diagnoses implementation errors in Lean formalizations and assesses their faithfulness to the corresponding NL mathematical statements via LLMs. We take a holistic view of autoformalization, identifying errors at a fine-grained level and associating them with inherent layers of the formalization process. Our approach iteratively detects and corrects syntactic, semantic, and grounding errors, including failures to faithfully capture the intended NL specification. The approach differs from prior work in that we disentangle error types, that we correct errors based in a hierarchical manner, and that we consider the grounding of the symbols in the raw text.

Complementary approaches target particular aspects of generation and refinement. Constrained generation enforces syntactic or grammatical requirements during decoding (Zhang et al., 2024b), while process supervision and process reward models provide feedback on intermediate mathematical reasoning (Lightman et al., 2024; Zhang et al., 2024a). Reinforcement learning with prover-derived rewards has also been used to improve the translation of natural-language mathematical statements into Lean (Yu et al., 2026). Retrieval-augmented approaches instead provide relevant definitions, dependencies, and theorems to guide formalization (Lu et al., 2026; Zhang et al., 2026). Beyond mathematical formalization or a static natural language query, we evaluate this process across first-order logic Han et al. (2024b), Dependent Type Theory, and temporal annotated logic, extending autoformalization to the evolving, interaction-dependent setting of τ^3-bench.

3 Autoformalization Failure Modes

Our approach is inspired by a concept from the field semiotics, specifically, the triadic relation (Peirce, 1955) that includes a sign (representamen), the subject matter (object) it refers to, and the mental concept (interpretant) formed in the mind. By extension, we also look at formalizing natural language as triadic relation- the symbols, their grounding and formal semantics. Here, syntax specifies how symbols can be manipulated; semantics specifies what those symbols mean; and grounding explains how those meanings become connected to non symbolic experience, or the

ID	Failure
I-A	Malformed Symbols
I-A-i	Improper operator
I-B	Representation of Facts
I-B-i	Unsupported Facts
I-B-ii	Supported Facts

Table 1: Category I (Syntax) Failures

ID	Failure
II-A	Symbol Grounding Failures
II-A-i	Multiplicity of symbols
II-A-ii	Improper grounding
II-B	Retrieval Failures
II-B-i	False Retrieval
II-B-i-f	Symbol Forgotten
II-B-i-nf	Symbol Persistence
II-B-ii	Symbol ignorance

Table 2: Category II (Grounding) Failures

world rather than depending solely on an external interpreter (Harnad, 1990; Taddeo & Floridi, 2005; 2007). Accordingly, we categorize failure modes in autoformalization into three types. Although not exhaustive, this taxonomy provides a practical structure for our analysis. To this taxonomy, we add an adversarial sub-type of each error where the given input text is intentionally misleading.

3.1 Category I Failure (Symbol):

We look at the failure related the underlying formal syntax, defined as a vocabulary of admissible symbols and the rules governing their formation of logical expressions with symbols, independent from their meaning. We list the related failures in table 1.

Malformed Symbol. A common failure mode in autoformalization arises when a model generates symbols or expressions that are incompatible with the defined syntax (I-A), yielding malformed constructs such as `2:30-time` or `has(reservation)(123)`. For example, introducing quantifiers such as $\forall$ or $\exists$ when the target language is propositional logic. While many syntactic and type errors can be detected directly by solver's parser or type checkers (Putra et al., 2026), including invalid notation (e.g., `n!` instead of `fact n`), some failures can be more subtle, such as applying a prefix function as an infix operator or inventing nonexistent constructs (e.g., `linear f` to encode linearity) (Wu et al., 2022). Moreover, syntactically valid formalizations may compile while still misrepresenting the intended logical structure, including quantifiers, operators, or constraints (Wang et al., 2026). We refer to an important subclass of these failures as *improper operator* errors (I-A-i), in which the model applies a logical operator incorrectly—for example, substituting disjunction for conjunction, omitting a required negation, or placing negation at the wrong scope.

Representation of Facts This type of error (I-B) concerns whether facts expressed in the NL input are faithfully represented in symbolic form, potentially reflecting ambiguous world knowledge of the model. In this paper, we will use "fact" to refer to any atomic primitive in the formalization. We consider two cases. First, *unsupported facts* (I-B-i) occur when the model introduces information that is not supported by the input. For example, the model may introduce a fact such as `student(:@alien, university)` without corresponding evidence from the input. We classify unsupported facts as soundness errors. In contrast, the second error type is *supported facts* (I-B-ii). Here, the error occurs when a fact is supported by the NL input but is has a discordant encoding or is incorrect in the underlying syntax. Such incongruity may arise from variations in required predicate arity, such as `has-reservation-insurance` as opposed to `reservation-insurance(yes)`, or from formatting differences such as `reservationflight` vs `ReservationFLIGHT`, or structural errors like unmatched parentheses. Moreover, small changes in logical structure can alter the intended meaning of a formalization even when it remains syntactically valid (Yang et al., 2025).

3.2 Category II Failure (Grounding):

In this section, we consider failures that arise in establishing and preserving the intended relation between natural-language concepts and their symbolic representations. We use grounding to refer to the process by which symbols are connected to what they are intended to represent, rather than acquiring meaning solely through relations to other symbols (Harnad, 1990). The associated failure modes are in table 2. Thus, a formalization may be syntactically well formed while nevertheless failing to preserve the intended use of its symbols (Mensfelt et al., 2026).

Symbol Grounding Failures. Failures occur when NL concepts are not mapped correctly to their symbolic representations. We consider two types of failures. *Multiplicity of symbols* (II-A-i) occurs when the same concept is mapped to multiple symbols which might not be present in the syntax due to inherent world knowledge of the model. For example, `has-reservation-insurance` and `reservation-insurance` may be introduced to represent the same concept. This results in missing associations between otherwise equivalent representations, and is therefore a completeness error. *Improper grounding* (II-A-ii) occurs when a concept is mapped to an incorrect symbol, such as grounding a concept to an correct but unintended constant under the same domain, or wrong order of domains in the predicate like `student(X, university)` to `student(university, X)`. Although the individual symbols may be supported by the input, their grounding does not correspond to an intended interpretation.

Retrieval Failures. Retrieval failures occur when the model fails to correctly recall or maintain previously established logical claims during formalization. *False retrieval* (II-B-i) occurs when an incorrect or irrelevant symbol is retrieved from the NL text. For example, when multiple reservation identifiers are associated with the same user, the model may retrieve an identifier of a different reservation in the past. Since the retrieved symbol introduces an incorrect association into the current context, we classify false retrieval as a soundness error. We further distinguish two temporal failures where the model is formalizing sequence of texts. *Symbol forgetting* (II-B-i-f) occurs when a previously introduced symbol is required by the current context but is no longer retained, making it a completeness error. Conversely, *symbol persistence* (II-B-i-nf) occurs when a previously introduced symbol is retrieved even though it is no longer relevant to the current context. Finally, *symbol ignorance* (II-B-ii) occurs when the correct symbol is allowed in the syntax but the model fails to identify or recover it. Unlike false retrieval, which selects an incorrect symbol, symbol ignorance results in the absence of the required symbolic information.

3.3 Category III Failure (Formal Semantics):

In this section, we examine failure with respect to an underlying formal semantics. This is discussed in (Harnad, 1990) with respect to what he defines as a "symbol system" where there is an underlying semantics. Often this is a set of satisfying assignments to symbols representing atoms. For example, in logic programming, this is often an interpretation or world; in modal logic this is often a Kripke structure of MDP; with a set of math equations this could be identifying if there exists a solution. We note that in some cases Category III may not be relevant, as it may be unknown if the resulting natural language produced has the quality of a symbol system (and in fact, that may be the reason for the autofromalization in those applications). Table 3 lists the failures in this category.

ID	Failure
III-A	Inconsistency
III-A-i	Inconsistency with domain knowledge
III-A-ii	Self-inconsistency
III-B	Non-Entailment
III-B-i	Model output not entailed by domain knowledge
III-B-ii	Domain knowledge not fully entailed by the model

Table 3: Category III (Semantic) Failures

Inconsistency. This type of error (III-A) occurs when the output of the the model violates logical coherence. We look at this in two ways. First, (III-A-i) is with respect to some external domain knowledge in symbolic format. Examples has been previously studied include a LTL specification for robot control, constraints on a system of equations, or cyberseucrity policies expressed as rules (Li et al., 2019; Wang et al., 2023; Cholvy & Cuppens, 1997). The error occurs when there is no semantic structure that satisfies both the output of the model and the domain knowledge simultaneously. For example, a type III-A-i error arises when the domain knowledge specifies $\neg foo$ and the model returns $foo \wedge bar$. The second type is self-inconsistency. Here the model produces a formalism for which there is no possible semantic structure that satisfies everything the model produced.

Non-Entailment. This type of error (III-B) occurs when there is a required entailment relationship between the model output and domain knowledge. Intuitively, for the two subcategories of this type

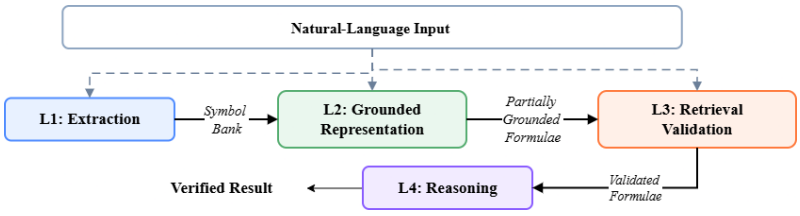


Figure 1: Reference architecture of the Layered AF protocol.

we have the requirement that either the model output should be logical consequence of the domain knowledge (III-B-i) or that the domain knowledge must be the logical consequence (III-B-ii). We would not expect that one would not seek to satisfy both of these requirements simultaneously. We would envision cases of III-B-i to arise when a large number of conclusions can be drawn from the domain knowledge, and model provides some additional insight from natural language to identify more narrowly-defined results. This type of relationship was recently examined with respect to software vulnerability analysis (Mukherji et al., 2026). Note that this is a stronger relationship than consistency as here *all* semantic structures satisfying the domain knowledge must also satisfy the model output to avoid the error. In the case of III-B-ii, we would require that all items in the domain knowledge must be concluded from the result of the model. An example use-case would be an inductive-style problem where we want the model to auto-formalize rules from text as a form on learning to account for some set of data (Cui & Chen, 2021; Bavikadi et al., 2025).

4 Reference Architecture for the Layered Protocol

We introduce a structured approach that decomposes autoformalization into distinct layers, enabling errors to be localized and addressed with layer-specific corrections. We assume an underlying formalism, $\mathcal{F}$, such as first-order logic, dependent type theory, or temporal annotated logic, that defines the available symbol classes, syntax, semantics, and inference mechanism governing the formal representation. We identify a conceptual set of sufficient 4-layers to standardize autoformalization as seen in Figure 1. Each layer has error detectors and correctors that are specific to formalism $\mathcal{F}$.

1. Extraction Layer: Given informal input, such as natural-language text or model output, the protocol begins by extracting formal entities that underlie the formal syntax. We introduce a structured, dynamically maintained *symbol bank* that represents and tracks an extracted set of constants, domains over constants, variables, and predicates together with the logical operators admitted by the underlying formalism. Each predicate is defined by its name, arity, and argument-domains. Here, we focus on the lowest-level syntactic errors for $\mathcal{F}$ (malformed symbols, I-A). This establishes the vocabulary available for subsequent formalization layers and enables statefulness, consistent symbol reuse, explicit grounding, and localized validation and correction.

2. Grounded Representation Layer: Using the symbol bank, this layer translates the informal input into formal expressions by grounding relevant information into a set of partially grounded formulae. While layer 1 focuses on identifying malformed symbols, layer 2 focuses on the quality of the facts and that symbol grounding was performed correctly (I-B-i, I-B-ii, II-A-i, II-A-ii). The resulting facts are consolidated into an evolving single data structure for downstream validation. This evolving formal representation is refined across layers.

3. Retrieval Validation Layer: While layer 2 looks to initially establish logical constructs, this layer ensures that the definition of the symbols is persistent, validating facts against their underlying evidence, including the original informal input and available environment or tool responses that can evolve with the text. The facts are cross-checked against the evidence to identify false retrievals, forgotten facts, stale facts, missing information (II-B-i, II-B-i-f, II-B-i-nf, II-B-ii), and updates the evolving formal representation.

4. Reasoning Layer: Finally, the reasoning layer evaluates the validated logical facts under the semantics of the underlying formalism. Depending on the reasoning task, this may involve external domain knowledge such as a rule set or set of constraints. In this layer, we can detect reasoning-level errors (III), the selection of which errors is application dependent. Considerations on the configuration of this layer include whether we expect the resulting formalism to be self-consistent, whether domain knowledge is present, and/or we expect a consistency vs. entailment relationship with the domain knowledge.

Algorithm 1 Layered AF

Require: Input x, ordered layers $\mathcal{L}$, initial representation z, initial symbol bank B
Ensure: Final formal representation z
  for each layer $i \in \mathcal{L}$ **do**
    $E_i \leftarrow \bigcup_{DETECTOR \in \mathcal{D}_i} DETECTOR(x, z, B)$ ▷ Detect errors associated with layer i
    for each error $(e, t) \in E_i$ **do** ▷ Iterate on each error (e) and its associated error type (t)
      $z \leftarrow CORRECTOR_t(x, z, B, e)$ ▷ Apply localized correction for error type t
    end for
  end for
  return z

Pipeline Description. We instantiate the verify-retry paradigm as a four-stage pipeline that incrementally autoformalizes informal data into a verifiable formal representation (Algorithm 1). At each layer, the protocol transforms the evolving representation z according to the underlying formalism, detects errors associated with that layer according to the taxonomy in Section 3, and applies the corresponding correction mechanisms. Each detected error e identifies both its taxonomy type and the affected portion of the current formal representation. The corresponding type-specific corrector operates only on this localized representation, preserving unaffected portions of z. We apply deterministic repairs where the intended correction is unambiguous—for example, canonicalizing malformed predicate names, correcting argument order and domain-type inconsistencies, resolving or rejecting arity mismatch, removing unsupported facts, carrying forward persistent facts, and removing facts invalidated by later reasoning steps or newer environment observations. For cases that cannot be resolved reliably from structural checks alone, such as ambiguous false retrievals or missing values, the repair is implemented by an LLM re-prompt using the relevant context, current formal representation, and available environment or solver feedback. Details are described in Appendix F,G,H. Finally, the reasoning layer performs inference over the current formal representation using a symbolic solver over the current formal representation and applicable rule set, or an LLM-based substitute.

5 Results and Discussion

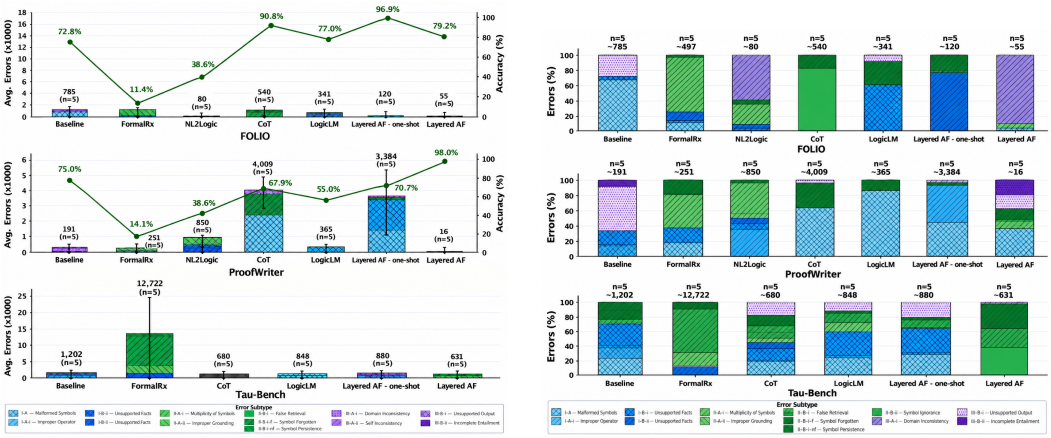
We evaluate on three datasets spanning deductive reasoning, and an application of agentic tool use givering a set of somain rules, with different underlying logical formalisms. FOLIO (Han et al., 2024a) is an expert-annotated benchmark designed to evaluate natural-language reasoning with first-order logic. We use the full train split of 1001 instances, each consisting of NL premises and conclusions whose logical status is labeled True, False, or Uncertain. ProofWriter (Tafjord et al., 2021) is a deductive reasoning benchmark in which models determine the entailment status of a query given a set of NL facts and rules. We use the full 600-instance development split under the open-world assumption at reasoning depth of 5, balanced across the labels- True, False, and Unknown. τ^3-bench (Yao et al., 2024) evaluates agentic tool use in multi-turn interactions under a NL domain-specific policy governing tool use. We use all 50 airline-domain tasks run with Claude Opus 4.8 (as both the LLM simulated user and the agent) for four independent trials per task, yielding 200 conversation trajectories. Each trajectory consists of a complete conversation ranging from 8 to 65 messages between a LLM simulated user and LLM-based tool-calling agent, together with the domain policy. We examine six methods on the datasets, described as follows. *One-shot Baseline (OS-Baseline)* is a single LLM call that reads the natural-language problem and emits the answer directly (for FOLIO, the premise/conclusion formulas together with a stated True/False/Uncertain status). Note that a solver is not used. *FormalRx* (Wang et al., 2026) is a formalization baseline prompted with the structured-cause-of-inconsistency (SCI) taxonomy that translates the problem into formal representations, which are then verified by the domain solver (Lean/Z3/ PyReason), with misalignments mapped to taxonomy categories. *NL2Logic* (Putra et al., 2026) is a grammar-guided method that recursively parses each natural-language statement into first-order logic and executable solver code (Z3 for FOLIO, Lean for ProofWriter), then checks entailment. *Logic-LM* (Pan et al., 2023). Translates natural-language problems into formal logic representations, executes them with a solver, and uses the solver output to produce the final answer. *Layered AF one-shot (LAF-OS)* is included

as an ablation of our approach. It collapses the LLM-calling layers (L1–L4) into a single combined extraction call and then runs the solver on the resulting formalization. It isolates the effects of layer decomposition and correction while preserving formal reasoning. *Layered Autoformalization (LAF)* if our full neurosymbolic pipeline decomposes the problem into four layers — with per-layer correctors that detect and repair upstream errors before the formalization is passed to a symbolic solver (Lean 4 (Moura & Ullrich, 2021), Z3 (De Moura & Bjørner, 2008), or PyReason (Aditya et al., 2023) depending on the domain). The final reasoning status comes from the solver, not the LLM. *MATP* (Zheng et al., 2026) in a multi-agent theorem-proving method that generates intermediate reasoning steps and verifies them with the Vampire theorem prover. Note that they cannot handle Uncertain as an entailment query that is inherent in FOLIO (run on 677 instances), Proofwriter (run on 344 instances). So, we only report and compare numbers on non-uncertain instances. Note that MATP and NL2Logic are not directly applicable to τ^3-bench, as they formulate reasoning over a fixed set of facts rather than evolving interaction states, so we omit them for those tasks. In τ^3-bench, facts and actions are introduced across turns, requiring temporal state tracking and reasoning over dynamically updated information like in Layered AF.

Metrics. We report per-category errors introduced in the Section 3. From this taxonomy we further label errors as 'soundness' and 'completeness'. Here, soundness errors are those when the system assert an unsupported conclusion (I-B-i, II-A-ii, II-B-i, II-B-i-nf) and completeness errors are when it misses a valid one (I-A, I-B-ii, II-A-i, II-B-i-f, II-B-ii, III). Lower numbers are better. Similarly, we look at task accuracy as the fraction of instances whose final entailment status matches the gold label in the FOLIO and Proofwriter dataset. For the methods using a formal solver, we additionally record the executable rate, the fraction of problems where the generated formal representation compiles in the solver, and executable accuracy, the fraction correct among those that executed — separating formalization failures from reasoning errors. For all our experiments, we use a high-memory compute node, a Giga Computing R163-Z34 server with a single-socket AMD EPYC 9755 128-Core Processor (256 threads) and 768 GB RAM. All LLM inference is performed via a remote API (AWS Bedrock). Every method is run under multiple LLMs including GPT-5.6-sol, Claude Opus-5, Claude Haiku 4.5, Qwen 3 80B Next, and Llama 4 Maverick. Symbolic reasoning engines used are Lean 4, Vampire, Z3, and PyReason. Code available at https://anonymous.4open.science/r/anonymous-submission-iclr-7AE7/README.md.

Autoformalization Error Breakdown. We examine the total errors in each method relative to the baselines. Figure 2a shows that LAF consistently reduces the average errors across all three datasets. On FOLIO, LAF reduces errors across five LLMs over OS-baseline by 93.0%, outperforming all baselines and consistently reducing errors across all five LLMs (87.2%–95.3%). Among the baselines, NL2Logic comes closest (89.8%), while the remaining methods yield substantially smaller reductions (31.2%–56.5%). Moreover, on the non-uncertain instances for which MATP is applicable, LAF outperforms MATP as well (MATP increases errors by 1340.9% while LAF reduces errors by 99.7%). Further comparison with MATP is in Appendix E. On ProofWriter, LAF reduces the average errors by 91.6%, while the other methods increase average errors by 31.3%–2001.4%. Using LAF provided better error reduction across all individual LLMs (see Appendix A) with the exception of GPT-5.6, where OS-basleine has only 1 error, compared with 12 for LAF. However, GPT-5.6 achieves 99.83% accuracy under OS-baseline, indicating that the benchmark is nearly saturated for this model. Furthermore, on the non-uncertain instances, we see a reduction in errors for MATP over OS-baseline of 21.1% while LAF outperformed MATP by 89.5%. On τ^3-bench, LAF reduces the average errors by 47.6%. The reduction is observed across all five LLMs, ranging from 28.3% to 66.7%. This compares favorable when compared to 43.5% for COT and contrasts with a increase in average errors by 958.03% with FormalRx. This suggests that the improvement extends beyond logical reasoning benchmarks to the multi-turn setting of τ^3-bench, rather than being specific to a model or dataset.

We also examined task accuracy (complete by-LLM results in Appendix B), reproducing comparable accuracy numbers as described in papers for MATP (Zheng et al., 2026), NL2Logic (Putra et al., 2026), and Logic-LM (Pan et al., 2023). We found that LAF improves accuracy by 8.73% over OS-baseline on FOLIO as seen in figure 2. Notably, CoT, which was not evaluated in the original work, achieves highest task accuracy, outperforming LAF on FOLIO for four of five LLMs. However, this advantage does not extend to errors- it has $10\times$ more, where LAF consistently performs better. Thus, accuracy alone can obscure substantial differences in underlying error burden. Our error analysis complements the accuracy results, illustrating that LAF's advantage in reducing



(a) Average number of errors with task accuracy. (b) Error distribution.

Figure 2: Error analysis across all models and 3 datasets for each taxonomy type.

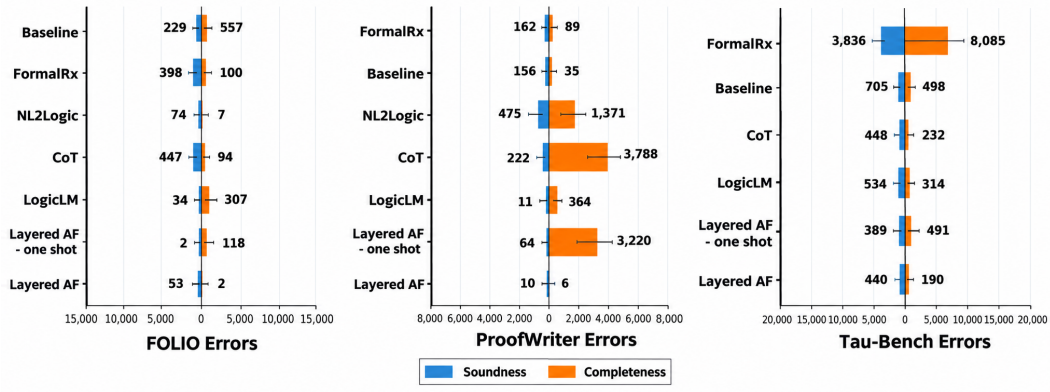


Figure 3: Soundness and Completeness errors averaged across all LLMs.

formalization errors. On ProofWriter, we observe that LAF improves task accuracy by 30.57% over the OS-baseline and obtains the best result.

Error Type Analysis. We investigate the composition of the errors in Figure 2b. On FOLIO, OS-baseline errors consist predominantly of malformed responses (I), followed by logical inconsistencies (III). LAF reduces overall errors by 93.0% and nearly eliminates Category I errors. On ProofWriter, the OS-baseline is dominated by inconsistencies (III), which LAF nearly eliminates. Overall errors decrease by 91.6%, with the remaining errors distributed across inconsistencies (III) and malformed responses (I). We find that, as Category I and II errors are substantially reduced, Category III errors constitute a larger share of the residual errors. Finally, on τ^3-bench, LAF reduces overall errors by 47.6% while completely eliminating Category I errors, including all unsupported-fact errors (I-B-i). The remaining errors are predominantly false-retrieval failures (92.2%), due to stale symbols, and symbol forgetting. Thus, the residual failures on τ^3-bench primarily concern retrieving and maintaining information across multiple conversation turns.

Soundness/Completeness Analysis. We further analyze the soundness or completeness failures of the errors as defined in section 3. On FOLIO, LAF yields over $14\times$ fewer completeness errors than the OS-baseline while maintaining near-zero soundness errors. In contrast, although CoT and FormalRx reduce completeness errors, they introduce hundreds of soundness errors. On ProofWriter, LAF similarly yields nearly $12\times$ fewer completeness errors, while also reducing soundness errors over baseline. Finally, on τ^3-bench, LAF reduces both soundness and completeness errors. On τ^3-bench, LAF reduces soundness and completeness errors by $1.5\times$ on average. Overall, these results suggest that LAF's reduction in total errors does not come from trading soundness for completeness and reduces overall soundness errors.

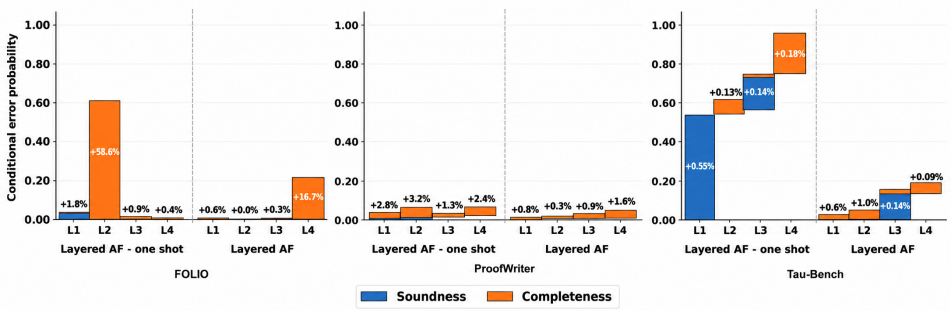


Figure 4: Conditional probability on each layer in reference architecture.

Component-wise Study of the Reference Architecture. We evaluate each LAF layer conditionally on the success of preceding layers (Algorithm 1), reporting $\mathrm{ErrorRate}(L_k) = \mathrm{Errors}(L_k)/(\mathrm{facts}(L_k) + \mathrm{symbols}(L_k))$. On FOLIO, Layers 1–3 are nearly error-free, with errors concentrated in L4; reasoning models achieve substantially lower L4 error rates (3.0–5.9%) than non-reasoning models (29.2–37.0%). On ProofWriter, errors remain low throughout the pipeline, with grounding and reasoning error rates below 2% across all five LLMs. In contrast, τ^3-bench shifts the bottleneck to retrieval validation (L3), where error rates range from 4.7% to 21.8%, while earlier layers remain nearly error-free. This is consistent with LAF's error decomposition, where 92.2% of the remaining τ^3-bench errors belong to the false-retrieval category.

Case Study 1. From τ^3bench, to show how LAF prevents previously established information from being lost during multi-turn formalization. The conversation initially establishes $\mathsf{HasTravelInsurance}(r)$, $\mathsf{CancellationReasonHealth}(r)$, but both facts are omitted in subsequent turns despite no evidence invalidating them, resulting in a Category II-B-ii missing-fact error. Our detector identifies the discrepancy, and the deterministic corrector restores both facts with their temporal provenance. This correction directly affects downstream reasoning: the cancellation policy requires $\mathsf{CanCancel}(r) \leftarrow \mathsf{CancelReservation}(r) \wedge \cdots \wedge \mathsf{HasTravelInsurance}(r) \wedge \mathsf{CancellationReasonHealth}(r)$ Before correction, only six of eight required conditions are present and the rule does not fire. After restoring the missing facts, all conditions are satisfied, deriving can_cancel(Anya_Garcia_reservation) . Thus, correcting the missing facts changes the downstream reasoning outcome and prevents an eligible cancellation from being incorrectly treated as ineligible.

Case Study 2. Here LAF ensures that extracted facts are grounded to the canonical entities defined by the symbolic representation. Grounded representation layer initially extracts has_booking_user('Ivan Rossi', "Ivan Rossi's upcoming May 21 reservation") is_flight_change("Ivan Rossi's upcoming May 21 reservation"). However, the conversation provides the user and reservation identifiers ivan_rossi_8555 and OWZ4XL. Retrival Validation detects this Category II-B-i grounding error by checking predicate arguments against the entity domains specified by the symbol bank. An LLM corrector maps the descriptive references to their entities: has_booking_user('ivan_rossi_8555','OWZ4XL'), is_flight_change('OWZ4XL'). This correction ensures that predicates referring to the same reservation consistently bind to r = OWZ4XL, allowing detection of violation to such as $\mathsf{violation_modify_basic_economy_flights}(r) \leftarrow \mathsf{update_reservation_flights}(r) \wedge \mathsf{is_flight_change}(r) \wedge \mathsf{is_basic_economy}(r)$.

6 Conclusion

We introduce Layered Autoformalization (LAF), a structured approach for identifying and correcting failures through a hierarchical taxonomy of syntax, grounding, and semantic errors. Across FOLIO, ProofWriter, and τ^3-bench, LAF reduces average formalization errors by 93.0%, 91.6%, and 47.6%, respectively. These results demonstrate the significance of localizing and correcting errors at the layer where they arise during autoformalization under different formalism. There are several directions for future work. As this framework is inherently modular, one direction for future work would be to incorporate other techniques error taxonomy and layered architecture. We also think that the symbol bank is an important structure warranting further study, as it could more readily allow agents to create their own formal representations. Finally, we believe this method could also enable improved agentic collaboration, with agents sharing formal representations, perhaps via a common symbol bank.

AI USE STATEMENT

In this work, we used generative AI tools for code generation and debugging, data analysis and visualization, and language editing and refinement of manuscript text. We have not used generative AI tools to autonomously conduct experiments, determine research conclusions, or generate unverified experimental results, and other disclosure categories not relevant to this work are not applicable. Additionally, we used generative AI tools to assist with formatting and improving the clarity and presentation of technical content. We reviewed all AI-assisted work. AI-generated code was inspected and tested against the experimental data, generated figures and quantitative results were verified against the underlying outputs, and AI-assisted manuscript text was reviewed and revised by the authors for technical accuracy and consistency with the experimental findings. All reported results and conclusions were determined from the authors' experiments and analyses rather than from generative AI outputs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We plan to release, upon acceptance, the full pipeline, our implementations of the seven comparison methods, the taxonomy definitions, and the per-run artefacts behind every table — per-layer error records, symbol banks, and timing logs for all three benchmarks and five models.

Four considerations limit absolute reproducibility. **First**, all five models were accessed through a managed inference API and three are closed-source, so our exact configuration becomes unavailable once those versions are deprecated; the two open-weight models can be re-hosted. **Second**, the API exposes no random seed, and the results might vary with the `temperature` field, so decoding is left at provider defaults. **Third**, we repeat each τ^3-bench task 4 times, but FOLIO (1001 items) and ProofWriter (600 items) are single-pass: the $\pm$ figures we report are the spread across the five models, not run-to-run variance, which we do not characterize. **Fourth**, the symbolic back-ends are floor-pinned (PyReason $\geq$ 2.0.0, Z3 $\geq$ 4.12.0) and the Lean 4 and Vampire binaries sit outside the Python environment, so a newer solver could change the L5 verdicts our reasoning-layer numbers rest on.

REFERENCES

Dyuman Aditya, Kaustuv Mukherji, Srikar Balasubramanian, Abhiraj Chaudhary, and Paulo Shakarian. PyReason: Software for open world temporal logic. In *AAAI Spring Symposium*, 2023.

Divyagna Bavikadi, Dyuman Aditya, Devendra Parkar, Paulo Shakarian, Graham Mueller, Chad Parvis, and Gerardo I. Simari. Geospatial trajectory generation via efficient abduction: Deployment for independent testing. *Electronic Proceedings in Theoretical Computer Science*, 416: 274–287, February 2025. ISSN 2075-2180. doi: 10.4204/eptcs.416.24. URL http://dx.doi.org/10.4204/EPTCS.416.24.

Laurence Cholvy and Frédéric Cuppens. Analyzing consistency of security policies. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp. 103–112. IEEE, 1997. doi: 10.1109/SECPRI.1997.601324.

Wanyun Cui and Xingran Chen. Open rule induction. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pp. 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3540787992.

William H. English, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. Grammar-forced translation of natural language to temporal logic using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 15370–15383, 2025.

Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong. Herald: A natural language annotated lean 4 dataset. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Se6MgCtRhz.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander R. Fabbri, Wojciech Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. Folio: Natural language reasoning with first-order logic, 2024a. URL https://arxiv.org/abs/2209.00840.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander Fabbri, Wojciech Maciej Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22017–22031, Miami, Florida, USA, 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1229. URL https://aclanthology.org/2024.emnlp-main.1229/.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3): 335–346, 1990. ISSN 0167-2789. doi: 10.1016/0167-2789(90)90087-6. URL http://dx.doi.org/10.1016/0167-2789(90)90087-6.

Seongjae Kang, Taehyung Yu, and Sung Ju Hwang. Policyguard: A dialogue-grounded sub-agent verifier for policy adherence in llm agents, 2026. URL https://arxiv.org/abs/2606.29225.

Chenyi Li, Wanli Ma, Zichen Wang, and Zaiwen Wen. Sita: A framework for structure-to-instance theorem autoformalization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 19224–19232, 2026. doi: 10.1609/aaai.v40i23.38997.

Jianwen Li, Moshe Y. Vardi, and Kristin Y. Rozier. Satisfiability checking for mission-time LTL. In *Computer Aided Verification*, pp. 3–22. Springer, 2019. doi: 10.1007/978-3-030-25543-5_1.

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

Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding complex natural language commands for temporal tasks in unseen environments. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1084–1110, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

Wangyue Lu, Lun Du, Sirui Li, Ke Weng, Haozhe Sun, Hengyu Liu, Minghe Yu, Tiancheng Zhang, and Ge Yu. Automated formalization via conceptual retrieval-augmented llms. In *International Conference on Learning Representations*, 2026.

Lezhi Ma, Shangqing Liu, Yi Li, Xiaofei Xie, and Lei Bu. Specgen: Automated generation of formal program specifications via large language models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 16–28. IEEE, 2025. doi: 10.1109/ICSE55347.2025.00129.

Agnieszka Mensfelt, David Tena Cucala, Santiago Franco, Angeliki Koutsoukou-Argyraki, Vince Trencsenyi, and Kostas Stathis. Towards a common framework for autoformalization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 40971–40980, 2026. doi: 10.1609/aaai.v40i48.42132.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, pp. 625–635, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-79875-8. doi: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

Kaustuv Mukherji, Jaikrishna Manojkumar Patil, Colton Payne, Paulo Shakarian, Dana Warmsley, Nigel Stepp, and Evelyn Kim. Entaillm: Verifying llm-generated vulnerability discovery paths with domain knowledge via logic programming, 2026. URL https://arxiv.org/abs/2608.01763.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.248.

Yu Pei, Yongping Du, and Xingnan Jin. Fover: First-order logic verification for natural language reasoning. *Transactions of the Association for Computational Linguistics*, 13:1340–1359, 2025. doi: 10.1162/tacl.a.41.

Charles Sanders Peirce. Logic as semiotic: The theory of signs. In Justus Buchler (ed.), *Philosophical Writings of Peirce*, pp. 98–119. Dover Publications, New York, 1955. Original work written c. 1897.

Rizky Ramadhana Putra, Raihan Sultan Pasha Basuki, Yutong Cheng, and Peng Gao. Nl2logic: Ast-guided translation of natural language into first-order logic with large language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6035–6051. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-eacl.317. URL http://dx.doi.org/10.18653/v1/2026.findings-eacl.317.

Damien Sileo. Scaling synthetic logical reasoning datasets with context-sensitive declarative grammars. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5275–5283, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.301. URL https://aclanthology.org/2024.emnlp-main.301/.

Mariarosaria Taddeo and Luciano Floridi. Solving the symbol grounding problem: a critical review of fifteen years of research. *Journal of Experimental Theoretical Artificial Intelligence*, 17:419–445, 12 2005. doi: 10.1080/09528130500284053.

Mariarosaria Taddeo and Luciano Floridi. A praxical solution of the symbol grounding problem. *Minds and Machines*, 17(4):369–389, 2007. doi: 10.1007/s11023-007-9081-3.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language, 2021. URL https://arxiv.org/abs/2012.13048.

Haocheng Wang, Baiyu Huang, Yingjia Wan, Xiao Zhu, Xiaoyang Liu, Yinya Huang, and Zhijiang Guo. Formalrx: Rectify and examine semantic failures in autoformalization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Zifan Wang, Saranya Vijayakumar, Kaiji Lu, Vijay Ganesh, Somesh Jha, and Matt Fredrikson. Grounding neural inference with satisfiability modulo theories. In *Advances in Neural Information Processing Systems*, volume 36, pp. 22794–22806, 2023.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Yutong Wu, Di Huang, Ruosi Wan, Yue Peng, Shijie Shang, Chenrui Cao, Lei Qi, Rui Zhang, Xishan Zhang, Zidong Du, Jie Yan, and Xing Hu. Stepfun-formalizer: Unlocking the autoformalization potential of llms through knowledge-reasoning fusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33980–33988, 2026. doi: 10.1609/aaai.v40i40.40691.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin E. Lauter, Swarat Chaudhuri, and Dawn Song. Position: Formal mathematical reasoning—a new frontier in AI. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 82384–82398. PMLR, 2025.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the power of large language models for natural language to first-order logic translation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6942–6959, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.375. URL https://aclanthology.org/2024.acl-long.375/.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Zhe Ye, Zhengxu Yan, Jingxuan He, Timothe Kasriel, Kaiyu Yang, and Dawn Song. VERINA: Benchmarking verifiable code generation. In *International Conference on Learning Representations*, 2026.

Xuejun Yu, Jianyuan Zhong, Zijin Feng, Pengyi Zhai, Roozbeh Mohit, Wei Ng, Haoxiong Liu, Ziyi Shou, Jing Xiong, Yudong Zhou, Claudia Ong, Austen Sugiarto, Yaoxi Zhang, Wai Ming Tai, Huan Cao, Dongcai Lu, Jiacheng Sun, Qiang Xu, Shen Xin, and Zhenguo Li. Mathesis: Towards formal theorem proving from natural languages. In *International Conference on Learning Representations*, 2026.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: Llm self-training via process reward guided tree search. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

Meiru Zhang, Philipp Borchert, Milan Gritta, and Gerasimos Lampouras. DRIFT: Decompose, retrieve, illustrate, then formalize theorems. In *International Conference on Learning Representations*, 2026.

Xinyi Zheng, Ningke Li, Xiaokun Luan, Kailong Wang, Ling Shi, Meng Sun, and Haoyu Wang. Beyond correctness: Exposing llm-generated logical flaws in reasoning via multi-step automated theorem proving. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2026.