AGENTIFIED ASSESSMENT OF LOGICAL REASONING AGENTS

Zhiyu Ni*
University of California, Berkeley
zhiyuni@berkeley.edu

Yifeng Xiao*
University of California, Berkeley
yifeng_xiao@berkeley.edu

Zheng Liang*
University of California, Berkeley
zhliang@berkeley.edu

ABSTRACT

We present a framework for evaluating and benchmarking logical reasoning agents when assessment itself must be reproducible, auditable, and robust to execution failures. Building on agentified assessment, we use an assessor agent to issue tasks, enforce execution budgets, parse outputs, and record structured failure types, while the agent under test only needs to expose a standardized agent-to-agent interface. As a case study, we benchmark an auto-formalization agent for first-order logic (FOL) reasoning on a solver-verified, repaired, and fully reviewed refined split of FOLIO. The agent translates natural language premises and conclusions into executable Z3Py programs and employs satisfiability modulo theories (SMT) solving to determine logical entailment. On the refined FOLIO validation set, the auto-formalization agent achieves 86.21% accuracy under the assessor protocol, outperforming a chain-of-thought baseline (71.92%). Recomputing against the original FOLIO validation labels yields 85.71% and 71.43%, respectively, because the original and refined validation splits contain identical premise-conclusion texts and differ only on three labels. Code is available at `https://github.com/zyni2001/agentified-logical-reasoning`.

1 INTRODUCTION

Evaluating and benchmarking reasoning agents is challenging because failures occur at multiple layers, including model reasoning and tool execution. As a result, static evaluation harnesses often conflate operational failures (e.g., timeouts, runtime errors, output parsing failures) with reasoning errors, and may hide these failure modes behind a single accuracy number. Moreover, traditional setups tightly couple benchmark logic to agent implementations, so integration effort grows with the number of benchmarks.

In this work, we study first-order logic (FOL) inference on FOLIO (Han et al., 2024) and adopt agentified agent assessment (AAA) as described in AgentBeats (AgentBeats, 2026), where assessment logic is implemented as an assessor agent that interacts with an agent under test through a standardized agent-to-agent (A2A) interface (A2A Protocol, 2026).

We conduct verification and repair on the FOLIO dataset (Han et al., 2024) using a data cleaning pipeline, establishing a more reliable benchmark. On the refined benchmark, we implement a white-box auto-formalization agent that achieves 86.21% accuracy on the fully reviewed FOLIO validation split, outperforming a chain-of-thought baseline (71.92%) (Wei et al., 2022).

*Equal contribution.

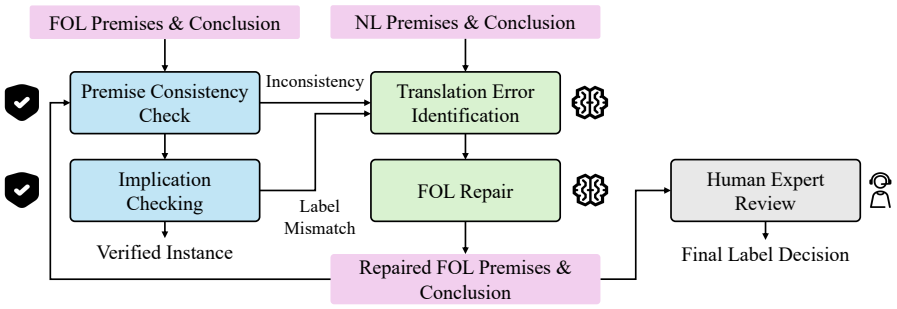

Figure 1: Overview of the data cleaning pipeline.

2 Benchmark and Data Cleaning

We evaluate on FOLIO (Han et al., 2024), a first-order logic (FOL) reasoning benchmark consisting of natural-language premises and conclusions paired with formal FOL annotations. FOLIO is derived from Wikipedia-based real-world scenarios and requires both natural language understanding and logical reasoning. For each example, FOLIO provides one of three labels: TRUE (the conclusion is logically entailed by the premises), FALSE (the conclusion contradicts the premises), or UNCERTAIN (the conclusion cannot be determined from the premises). We evaluate classification accuracy against human-annotated ground-truth labels.

2.1 Verification and Repair Pipeline

The original FOLIO dataset exhibits potential label errors and misalignments between natural language and formal annotations due to the complexity of semantic parsing. To establish a more reliable benchmark, we implement a systematic data cleaning pipeline (Figure 1) that leverages symbolic verification using the Vampire theorem prover (Kovács & Voronkov, 2013) to execute formal reasoning on FOL representations. When verification results conflict with expected labels, we identify and iteratively repair translation errors.

Given a premise set $\{\phi_1, \phi_2, \dots\}$ and a conclusion φ, we detail the pipeline as follows:

Formal Premise-Conclusion Checking. We verify the implication relation between premises and conclusion in two steps. First, we check whether the premises are consistent (i.e., whether $\bigwedge_i \phi_i$ is satisfiable). Then, we perform implication checking using the following satisfiability conditions:

$$\text{TRUE} \iff \bigwedge_i \phi_i \to \varphi \iff \bigwedge_i \phi_i \wedge \neg\varphi \text{ is unsatisfiable,} \tag{1}$$

$$\text{FALSE} \iff \bigwedge_i \phi_i \to \neg\varphi \iff \bigwedge_i \phi_i \wedge \varphi \text{ is unsatisfiable.} \tag{2}$$

When both implication checks fail, we label the instance as UNCERTAIN. We then compare verification results against expected labels to identify potential annotation errors.

NL-FOL Misalignment Identification and Repair. When premises are inconsistent or verification results conflict with expected labels, we employ a check-and-fix procedure using two LLM-based agents. A *critique agent* diagnoses systematic translation errors (e.g., unbalanced parentheses, lexical typos, and naming convention inconsistencies) by analyzing the original natural language and FOL annotations. A *refiner agent* then executes targeted corrections. We iteratively re-verify the repaired FOL until the expected labels are achieved. If the number of iterations exceeds a predefined threshold without resolution, we flag the instance for manual review. For the validation split, all flagged cases are then manually reviewed and assigned final labels before benchmarking. We release the cleaned and repaired FOLIO split at `https://huggingface.co/datasets/yfxiao/folio-refined`.

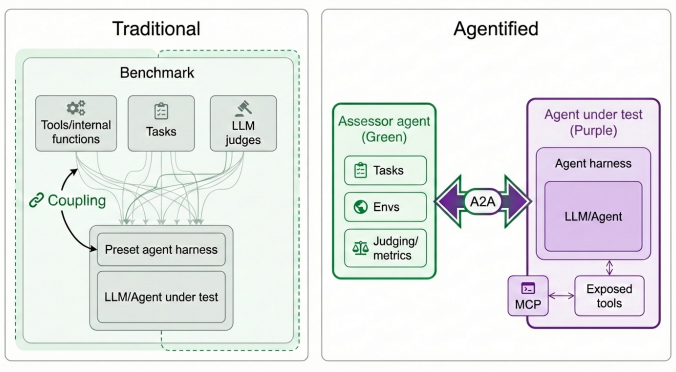

Figure 2: Traditional evaluation harnesses couple task execution, environments, and judging logic to a preset harness. In agentified assessment, an assessor agent evaluates an agent under test via an A2A interface (A2A Protocol, 2026), reducing integration overhead.

2.2 Benchmark Statistics

We classify instances into three categories: (i) verified directly with the original FOL, (ii) verified after automatic repair, and (iii) cases flagged for manual review. For the *training set* (1,001 examples): 674 (67.3%) verified directly, 23 (2.3%) after automatic repair, and 304 (30.4%) were flagged for manual review. For the *validation set* (203 examples): 154 (75.9%) verified directly, 10 (4.9%) after automatic repair, and 39 (19.2%) were flagged for manual review. These 39 validation cases were manually reviewed and assigned final labels before benchmarking, so the refined validation split used in our experiments contains all 203 examples. In total, we identify 3.8% potential label errors in the training set and 1.5% in the validation set, improving both the NL-FOL translation quality and the reliability of labels for evaluation.

3 Agentified Evaluation Framework

3.1 Agentified Assessment

We formalize the evaluation of reasoning agents by *treating assessment itself as an agent*. Instead of relying on a static evaluation script, we separate the system into two interacting components: an *agent under test*, which performs reasoning, and an *assessor agent*, which controls task execution, interprets outcomes, and assigns scores. We adopt the agentified agent assessment (AAA) abstraction described in AgentBeats (AgentBeats, 2026), where the assessor and the agent under test communicate through an agent-to-agent (A2A) interface (A2A Protocol, 2026).

The primary value of AAA is not to address individual operational issues (e.g., timeouts), which can also be handled in traditional benchmarks. Instead, it changes the integration cost model for benchmarking agents: in traditional setups, integration cost grows with the number of benchmarks ($O(n)$), whereas under AAA, an agent implements A2A once and can participate in many assessors ($O(1)$). This decoupling enables plug-and-play evaluation and architecture freedom for agents with diverse internal designs.

As illustrated in Figure 2, traditional benchmarks couple task orchestration and judging to a benchmark-specific harness. AAA decouples these components by packaging evaluation logic as an assessor agent that interacts with an agent under test through a standardized interface, allowing the assessor to evolve without breaking participants.

3.2 Assessor Agent (Evaluation Protocol)

The *assessor agent* implements the evaluation protocol. For each premise and conclusion pair, it issues the task to the agent under test, enforces execution budgets (e.g., timeouts and retries), parses

and validates the response, and assigns a final label (TRUE, FALSE, or UNCERTAIN) according to the benchmark criteria.

If the agent under test outputs additional text, the assessor parses the final label deterministically; non-parseable outputs are recorded as PARSEERROR. Rather than discarding failures, the assessor records structured failure types such as TIMEOUT, RUNTIMEERROR, and PARSEERROR. Finally, the assessor emits a machine-consumable evaluation artifact containing per-instance records (gold label, predicted label, correctness, error type if any, latency) and aggregate metrics.

3.3 REASONING AGENTS UNDER TEST

We benchmark two agents under the same assessor.

Chain-of-thought baseline. We use chain-of-thought prompting (Wei et al., 2022): the agent is instructed to reason step by step and then output its final answer as the last line containing exactly one label in {TRUE, FALSE, UNCERTAIN}, which is parsed by the assessor.

Auto-formalization agent. The auto-formalization agent solves first-order logic (FOL) inference problems by translating natural language premises and conclusions into executable symbolic programs and performing logical reasoning via solver execution (Pan et al., 2023; Ni et al., 2026). The reasoning process follows a two-stage pipeline. In Stage 1 (Code Generation), a language model generates executable Z3Py (De Moura & Bjørner, 2008) code from the input. In Stage 2 (Execution and Verification), the generated program is executed in a sandbox environment with a 60-second timeout.

Logical validity is determined through satisfiability checking as specified in the benchmark definition (Section 2): if $\bigwedge_i \phi_i \wedge \neg\varphi$ is unsatisfiable, the conclusion is labeled TRUE; if $\bigwedge_i \phi_i \wedge \varphi$ is unsatisfiable, it is labeled FALSE; otherwise, the label is UNCERTAIN.

To improve robustness, the agent incorporates a self-repair loop with up to three attempts. When execution fails due to syntax errors or malformed quantifiers, the agent extracts the error message and performs targeted code repair before re-execution.

Implementation note. We implement the assessor and agents under test on top of AgentBeats (AgentBeats, 2026), where they communicate via A2A (A2A Protocol, 2026) (and optionally expose tools via MCP (Anthropic, 2024)). We also constructed a logical reasoning leaderboard that runs the assessor against registered agents and records per-run artifacts (accuracy, latency, and failure types), enabling reproducible comparisons across agents.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP AND BASELINE

We evaluate on the refined FOLIO validation split comprising 203 examples after verification and review: 154 verified directly with the original FOL, 10 verified after automatic repair, and 39 manually reviewed cases assigned final labels before benchmarking. Both the chain-of-thought baseline agent and the auto-formalization agent use Gemini 2.5 Flash (Comanici et al., 2025) as the backbone LLM with a temperature $T = 0.0$ for deterministic outputs and default maximum output tokens $n = 65535$. Both agents use a system-user prompt structure. For the chain-of-thought baseline, the system prompt elicits step-by-step reasoning and requires the final answer label to appear as the last line; for auto-formalization, it includes Z3 code generation instructions with syntax guidelines. No in-context learning (ICL) examples are provided for either agent.

4.2 RESULTS AND ANALYSIS

Table 1 summarizes overall accuracy on both the refined and original FOLIO validation labels. The chain-of-thought baseline achieves 71.92% (146/203) on the refined split, while the auto-formalization agent improves accuracy to 86.21% (175/203). With an A2A timeout of 300s and concurrency 4, both runs complete all 203 cases without assessor-side parse errors, timeouts, or

uncaught runtime exceptions. The average wall-clock time per case is 8.80s for the baseline and 17.73s for auto-formalization.

To bridge the refined split back to the original FOLIO validation set, we compared the two released CSV files directly. They contain the same 203 premise-conclusion pairs, but three examples have different labels after verification and review. Therefore, we recompute original-split accuracy from the same predictions by re-scoring them against the original labels. This yields 71.43% (145/203) for the chain-of-thought baseline and 85.71% (174/203) for auto-formalization, showing that the refined-label correction changes the headline accuracy by only one example for each method.

Method	**Original**	**Refined**	**Parse**	**Timeout**	**Exception**
Chain-of-thought	145/203 = 71.43%	146/203 = 71.92%	0	0	0
Auto-formalization	174/203 = 85.71%	175/203 = 86.21%	0	0	0

Table 1: Overall results on original and refined FOLIO validation labels. The original-label scores are recomputed from the same prediction traces because the two validation files contain identical premise-conclusion texts and differ only on three labels. Parse/timeout/exception counts are assessor-side failure counts on the refined run.

Table 2 reports refined-split accuracy broken down by ground-truth label category. The largest improvement is observed in the FALSE category, where accuracy increases from 42.62% to 80.33%. Performance on TRUE cases is comparable across methods. The auto-formalization agent also improves on UNCERTAIN cases (81.16% to 91.30%), highlighting the advantage of solver-based reasoning in handling logical indeterminacy.

	Chain-of-Thought			**Auto-formalization**		
Category	Correct	Total	Acc.	Correct	Total	Acc.
True	64	73	87.67%	63	73	86.30%
False	26	61	42.62%	49	61	80.33%
Uncertain	56	69	81.16%	63	69	91.30%
Overall	146	203	71.92%	175	203	86.21%

Table 2: Per-category and overall accuracy on the refined FOLIO validation set.

Overall, these results demonstrate that formal verification improves robustness on this benchmark, leading to higher overall accuracy while preserving clean assessor-side execution traces.

5 CONCLUSION

This work benchmarks logical reasoning agents under agentified assessment, where an assessor agent enforces budgets, records structured failures, and emits auditable evaluation artifacts. As a case study, an auto-formalization agent that translates natural language problems into executable Z3Py code and uses solver execution outperforms a chain-of-thought baseline on both the refined FOLIO validation labels (86.21% vs. 71.92%) and the original validation labels recomputed from the same prediction traces (85.71% vs. 71.43%), with the largest gains on contradiction (FALSE) cases and consistent improvements on indeterminate (UNCERTAIN) cases. Future work could extend assessor policies and apply agentified assessment to richer tool-using agent settings.

REFERENCES

A2A Protocol. Agent2agent (a2a) protocol specification, 2026. https://a2a-protocol.org/latest/.

AgentBeats. Agentified agent assessment (aaa), 2026. https://docs.agentbeats.dev/.

Anthropic. Model context protocol (mcp), 2024. https://docs.anthropic.com/en/docs/mcp.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the

frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, et al. Folio: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22017–22031, 2024.

Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *International Conference on Computer Aided Verification*, pp. 1–35. Springer, 2013.

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

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

A Appendix: Reproducibility Details

Code and artifacts. The code implementation, evaluation scripts, and run artifacts are released at `https://github.com/zyni2001/agentified-logical-reasoning`. The refined FOLIO split used in our experiments is released at `https://huggingface.co/datasets/yfxiao/folio-refined`.

Prompt structure. Both agents use a system-user prompt format with Gemini 2.5 Flash at temperature 0.0 and no in-context examples. The chain-of-thought baseline prompt asks for step-by-step reasoning followed by a final line containing exactly one label in $\{$TRUE, FALSE, UNCERTAIN$\}$. The auto-formalization prompt asks the model to generate executable Z3Py code for the input premises and conclusion; when execution fails, a repair prompt is constructed from the original problem, the previous code, and the execution error.

Parsing and execution budgets. The assessor parses labels deterministically from the agent response and records PARSEERROR if no label can be recovered. The assessor uses an A2A request timeout of 300 seconds and evaluates up to four instances concurrently. The auto-formalization agent executes generated Z3Py code in a subprocess with a 60-second timeout and allows up to three repair attempts before returning UNCERTAIN.

Solver outcome mapping. Given premises $\{\phi_i\}$ and conclusion φ, the auto-formalization agent maps solver outcomes to benchmark labels as follows: if $\bigwedge_i \phi_i \wedge \neg\varphi$ is unsatisfiable, the answer is TRUE; if $\bigwedge_i \phi_i \wedge \varphi$ is unsatisfiable, the answer is FALSE; otherwise the answer is UNCERTAIN. The same three-way label space is used by the assessor when scoring both agents.

Original vs. refined validation labels. The original FOLIO validation CSV and the refined validation CSV contain the same 203 premise-conclusion pairs. They differ only on three labels after verification and manual review. Therefore, original-split accuracy is recomputed from the same prediction traces by rescoring them against the original labels, while refined-split accuracy is computed against the reviewed labels used in the main experiments.