GENERATION IS NOT SELECTION: NEURO-SYMBOLIC NATURAL LANGUAGE FORMALIZATION OF TEMPORAL LOGIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Natural language-to-temporal logic translation is challenging because a requirement can admit multiple plausible formalizations. We study whether neuro-symbolic systems can select the intended formalization when a correct candidate is already present. We present Pendulum, a pipeline separating neural candidate synthesis from symbolic analysis and final selection. Symbolic analysis provides solver-backed equivalence and behavioral evidence, which a downstream judge uses before deterministic finalization. We evaluate Pendulum on DS_1^{tricky}, a 306-instance future-time LTL dataset derived from the Little Tricky Logic benchmark. After auditing the benchmark, we evaluate on 258 determinate cases. A correct candidate is present for 92.2% of these cases, but the final system selects one correctly for 87.2%, yielding a 5.0-percentage-point coverage-to-selection gap. Controlled changes to synthesis, semantic analysis, and selection do not consistently eliminate this gap. Solver-backed behavioral evidence likewise does not reliably resolve selection when multiple semantically distinct candidates remain plausible. These results expose candidate coverage and candidate selection as distinct failure modes in neuro-symbolic formalization.

1 INTRODUCTION

A natural language requirement can admit multiple plausible formalizations Danso et al. (2026). Two formulas may be syntactically valid yet differ in temporal scope, operator choice, or the behaviors they permit Greenman et al. (2022). This creates a failure mode hidden by end-to-end translation accuracy: a system can generate the intended formalization, retain it among its candidates, and still return a different formula. In such cases, the failure is not generation. It is selection.

The distinction matters because formal analysis and formal selection answer different questions. Given candidate LTL formulas, a symbolic solver can determine equivalence, logical relationships, and distinguishing behaviors for non-equivalent candidates Geatti et al. (2021); Duret-Lutz et al. (2016). Such analyses expose semantic differences, but do not determine which non-equivalent formula expresses the intended reading of a natural language requirement. When multiple readings remain possible, the missing information is which interpretation the requirement was intended to encode Greenman et al. (2022).

Natural language-to-LTL translation has been studied extensively, including work examining syntactic validity and semantic correctness Danso et al. (2026); Pirozelli et al. (2024); English et al. (2025a). Existing systems combine neural generation with human, solver, or symbolic feedback to constrain, analyze, or refine generated specifications Cosler et al. (2023); Chen et al. (2023); Saha et al. (2026); Feng et al. (2025); Wang et al. (2025). These approaches demonstrate the value of formal analysis during generation. A complementary question remains: *when a correct formalization is already present among the candidates, can the system identify it as the intended one?*

This paper separates two measurable outcomes. Let $C(s)$ denote the candidate set generated for specification s, and let φ^* denote its intended formalization. *Candidate coverage* asks whether $C(s)$ contains a candidate semantically equivalent to φ^*; *candidate selection* asks whether the system ultimately commits to such a candidate. A system can have low coverage because synthesis fails to produce the intended formalization, or high coverage but lower selection because the correct

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

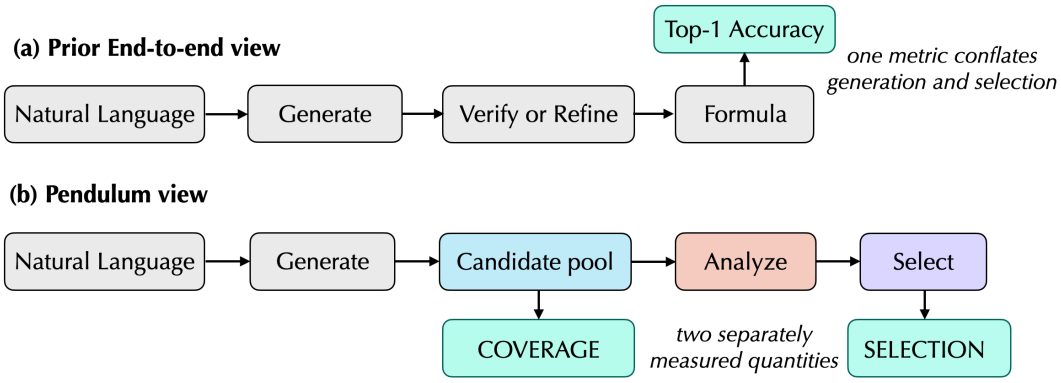


Figure 1: The measurement decomposition studied in this paper. End-to-end evaluation observes only the final formalization, conflating failures to generate a correct candidate with failures to select one. Pendulum measures candidate coverage over the generated pool separately from candidate selection at finalization.

candidate is available but another is chosen. Conventional top-1 accuracy collapses these cases into the same error. This distinction is illustrated in Figure 1.

This paper presents *Pendulum*, a fixed neuro-symbolic pipeline that makes this distinction explicit. Two synthesis paths produce a candidate pool; BLACK Geatti et al. (2021) provides solver-backed semantic and behavioral evidence; an LLM judge interprets this evidence in the context of the requirement; and deterministic finalization commits to one candidate. This separation enables selection to be measured when synthesis has already supplied a correct formalization.

The evaluation uses DS_1^{tricky}, the 306-instance future-time LTL dataset curated in Danso et al. (2026) from the *Little Tricky Logic* benchmark (Greenman et al., 2022). The dataset is distinct from *NL2LTL*, the natural language-to-LTL translation system of Fuggitti & Chakraborti (2023). We audit all 306 references because the coverage–selection distinction requires a well-defined correctness target. The audit identifies 55 confirmed reference errors and 48 ambiguous specifications, leaving 258 determinate cases. On these cases, the main configuration contains a correct candidate for 92.2% but selects one correctly for 87.2%, producing a 5.0-percentage-point coverage–selection gap. Controlled changes to candidate synthesis, semantic analysis, and selection do not consistently eliminate this gap; solver-backed behavioral evidence likewise does not reliably resolve selection among semantically distinct candidates.

The paper makes four contributions:

1. We formulate and empirically separate *candidate coverage* from *candidate selection* for natural language-to-LTL formalization, exposing failures that top-1 accuracy conflates.
2. We present and evaluate *Pendulum*, a fixed neuro-symbolic pipeline combining neural candidate synthesis, solver-backed semantic analysis, and explicit final selection.
3. We audit DS_1^{tricky}, identifying reference errors and separating ambiguous specifications from the determinate evaluation set.
4. We conduct controlled experiments characterizing the coverage–selection gap and the role of symbolic behavioral evidence in candidate selection.

Artifacts. All artifacts are available in the Github repository Anonymous (2026).

2 Pendulum: Neuro-Symbolic Formalization and Selection

We study natural language-to-LTL translation as a candidate-generation and selection problem. Pendulum separates candidate generation, semantic analysis, and final selection so that the availability of a correct candidate can be measured independently of the final decision.

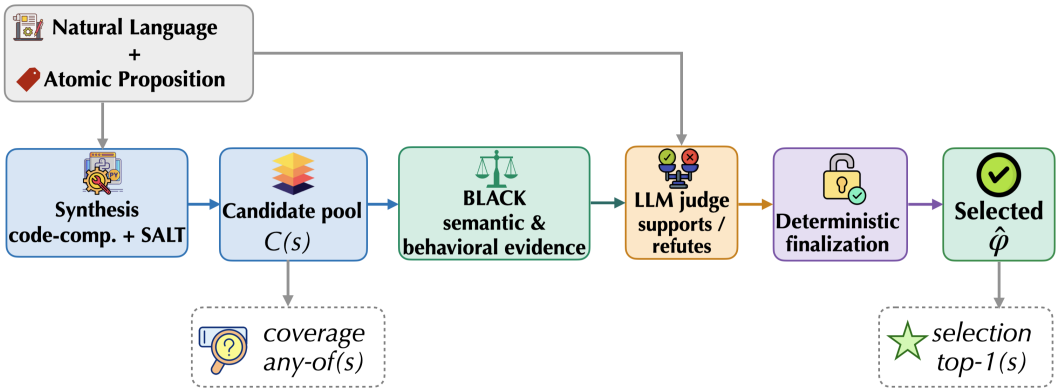


Figure 2: Pendulum separates candidate generation from final selection. Two synthesis paths produce a candidate pool, which BLACK analyzes semantically and behaviorally. An LLM judge combines this evidence with the requirement context, and deterministic finalization commits to one candidate. Candidate coverage is measured over the generated pool, while top-1 selection is measured from the committed candidate.

2.1 Problem Formulation

Given a natural language requirement s and, when applicable, an atomic proposition vocabulary V, Pendulum produces a candidate set $C(s)$ of LTL formulas and commits to one formula $\hat{\varphi}$. Let φ^* denote the intended formalization. Correctness is evaluated by solver-backed semantic equivalence rather than syntactic matching; for example, `G(p -> Fq)` and `G(!p | Fq)` are equivalent.

We measure two quantities. *Candidate coverage* indicates whether a correct formalization is present in the generated set:

$$any - of(s) = \mathbb{1}[\exists\, \varphi \in C(s) : \varphi \equiv \varphi^*]\,.$$

Candidate selection indicates whether the committed formula is correct:

$$top - 1(s) = \mathbb{1}[\hat{\varphi} \equiv \varphi^*]\,.$$

The difference between these quantities is the *coverage–selection gap*. Coverage therefore measures the correctness available to the selector, while selection measures the correctness ultimately returned by the system.

2.2 Candidate Analysis and Selection

Pendulum separates candidate synthesis, symbolic semantic analysis, and final selection. Candidate synthesizers first generate a pool of LTL formulas. The symbolic substrate, BLACK, then establishes semantic and behavioral relationships among candidates, including equivalence, entailment, consistency, and distinguishing behaviors. These analyses provide formal evidence about the candidate set but do not determine which candidate expresses the natural language requirement.

A downstream LLM judge evaluates the candidate formulas using the natural language requirement, atomic-proposition mapping, and BLACK's semantic and behavioral evidence. The resulting assessments are passed to a deterministic finalization stage that ranks the candidates and commits to one without an additional model call. Thus, symbolic analysis characterizes relationships among the formal alternatives, while the judge interprets those relationships in the context of the natural language requirement. When non-equivalent candidates correspond to different plausible readings, symbolic analysis can expose their behavioral differences without establishing which interpretation was intended.

Figure 2 summarizes the pipeline. The generation stage produces the candidate pool; BLACK analyzes the candidates and supplies semantic evidence; and the selection stage uses this evidence together with the requirement context to commit to one candidate. This separation enables candidate coverage to be measured independently of final selection, which we evaluate in §4.

Table 1: Defect taxonomy of the 55 confirmed erroneous references.

Error type	Count	Share of confirmed errors
Wrong temporal scope	36	65.5%
Wrong temporal operator	8	14.5%
Structural / other	4	7.3%
Wrong negation	3	5.5%
Wrong counting	2	3.6%
Dropped condition	1	1.8%
Other variable reference	1	1.8%
Total	**55**	**100%**

Implementation. Pendulum is implemented as a fixed state machine whose control flow is determined by code rather than model decisions. Atomic propositions are extracted once when not supplied, after which the code-completion and specification-synthesis paths generate candidates in parallel. The specification path produces a SALT representation that is compiled to LTL; compiler failures are handled within the bounded synthesis workflow. Candidates from both paths are parser-gated, merged, and deduplicated before semantic analysis. BLACK then produces the semantic and behavioral evidence supplied to the judge. Finalization deterministically ranks the candidate assessments and commits to the top-ranked formula without an additional model call. The surviving candidate set measures coverage, while the committed formula determines top-1 selection. Further pipeline details are given in Appendix F.2.

3 Benchmark Audit

The coverage–selection distinction depends on a well-defined correctness target. If a benchmark reference does not faithfully represent its natural language requirement, a system can be penalized for selecting a reasonable formalization, or credited for selecting an incorrect one. Because this ambiguity directly affects both coverage and selection, DS_1^{tricky} is audited before it is used for the primary evaluation.

3.1 Audit Procedure

The audit examines all 306 benchmark instances using automated syntactic and semantic checks, model-assisted screening, and author adjudication. The screening identifies potentially problematic references, including differences in temporal scope, operator choice, negation, counting, and proposition structure. These checks serve only to identify cases for review; they do not determine the final labels. Each flagged instance is examined against its natural language requirement, and the corresponding formulas are checked using solver-backed LTL semantics.

The audit distinguishes two cases. A reference is classified as *erroneous* when the natural language requirement supports a formalization that differs from the published reference. An instance is classified as *ambiguous* when the requirement itself admits multiple plausible formalizations and therefore does not provide sufficient evidence for a unique correctness target. Ambiguous instances are not forced into a single interpretation for the purpose of evaluation.

3.2 Audit Results

The audit identifies 72 instances whose published references require correction, of which 55 are confirmed erroneous. Separately, 48 instances are classified as ambiguous. Excluding the ambiguous instances from the primary correctness evaluation leaves 258 determinate specifications. The confirmed errors span multiple aspects of temporal specification, with incorrect temporal scope accounting for the largest category. The corrections are semantic rather than merely syntactic: changing temporal scope or operator choice can alter the behaviors described by a formula. For each accepted correction, the resulting formula is checked for syntactic validity, atomic-proposition consistency, and semantic non-equivalence to the published reference before it is used as an evaluation target.

The resulting 258 determinate specifications form the primary evaluation set. The ambiguous instances are retained as a separate partition rather than assigned forced labels. The original parseable benchmark is also retained as a historical reference point for comparison with results obtained before the audit; it is not used for the primary coverage and selection measurements.

The audit therefore defines the correctness target for the remainder of the paper. Candidate coverage is measured by semantic equivalence to the adjudicated reference, and candidate selection is measured by semantic equivalence of the ultimately committed candidate to that same reference. The audit establishes the validity of these measurements; it is not itself an evaluation of Pendulum.

4 EVALUATION

The evaluation separates two questions that conventional end-to-end translation accuracy conflates: whether a correct formalization is generated at all, and whether it is selected once available. We organize the evaluation around three research questions.

RQ1 (coverage). How often does candidate synthesis include a semantically correct formalization? This establishes the ceiling available to subsequent selection: a selector cannot recover a formalization that synthesis never produces.

RQ2 (selection). When a correct formalization is available, how reliably does the system select it? This isolates failures after candidate generation.

RQ3 (component effects). Do changes to semantic evidence, selection, or candidate construction reliably reduce the coverage–selection gap? We answer this with controlled interventions and regression checks to distinguish improvements from regressions.

4.1 EXPERIMENTAL SETUP

Benchmark and metrics. The primary evaluation uses the 258 determinate specifications obtained from the audit of DS_1^{tricky} in §3. Candidate correctness is determined by solver-backed logical equivalence rather than textual matching. For a specification s, candidate coverage $any-of$ is the fraction for which the candidate pool contains a formula semantically equivalent to the audited reference; top-1 selection $top-1$ is the fraction for which the formula ultimately committed by the system is equivalent to that reference. Their difference is the coverage–selection gap. We additionally retain the 303-instance parseable historical partition as a secondary anchor for comparison with pre-audit results; it is not used for the primary RQ1–RQ3 measurements.

Configuration. The headline configuration, used for the primary end-to-end results, uses a Codex synthesis pool and a Gemma-4 31B assessment model. Candidate generation, semantic analysis, and selection follow the fixed Pendulum control flow described in §2. BLACK provides solver-backed equivalence, entailment, consistency, and distinguishing-behavior evidence to the assessment stage; a deterministic ranker then commits to one candidate without an additional model call. The symbolic substrate and model seats are held fixed when individual components are varied. Complete configurations, prompts, and inference settings are given in Appendix A.

Controlled comparisons. For binary comparisons, we use paired exact McNemar tests and report the discordant counts. Intervention results are evaluated with a regression check: requirements newly solved and requirements newly broken are measured on a common denominator, and the net change is reported. This prevents an intervention from appearing beneficial merely because it solves a small number of additional cases while silently regressing on previously solved ones. Detailed statistical results are provided in Appendix C.

Contamination. DS_1^{tricky} was publicly available before the experiments, so pretraining exposure cannot be ruled out for the evaluated models. We therefore interpret the results as controlled comparisons on an audited benchmark rather than contamination-free estimates of generalization; this threat is discussed further in §7.

Table 2: Candidate coverage and top-1 selection when the synthesis pool and assessment model are varied. The coverage–selection gap is present in every configuration. Denominators differ slightly because a small number of specifications did not reach a terminal outcome in the Gemma-pool.

Configuration	any-of	top-1	Gap
Codex + Gemma-4 31B	**238/258 (92.2%)**	**225/258 (87.2%)**	**5.0 pp**
Codex + Gemma-4 12B	238/258 (92.2%)	221/258 (85.7%)	6.5 pp
Gemma + Gemma-4 31B	219/257 (85.2%)	197/257 (76.7%)	8.5 pp
Gemma + Gemma-4 12B	225/254 (88.6%)	193/254 (76.0%)	12.6 pp

4.2 RQ1: Candidate Coverage

RQ1 asks whether synthesis places a correct formalization within reach of the selection stage. For the main Codex synthesis pool, a semantically correct candidate is present for 237 of the 258 determinate specifications (91.9%) before downstream analysis and selection. The corresponding Gemma synthesis pool covers 219 of 258 specifications (84.9%). Thus, candidate availability is already high but remains an upstream limitation: no selection procedure operating on a given pool can recover a correct formalization that the pool does not contain.

After parser filtering, merging, and deduplication, the headline Pendulum pool contains a correct candidate for 238 of 258 specifications (92.2%). The small difference between the raw Codex pool and the final pool reflects candidate processing rather than a new synthesis model. The important point for RQ1 is that the correct formalization is available to the downstream selector for the large majority of the audited benchmark.

4.3 RQ2: Candidate Selection

RQ2 asks the central question: *when a correct formalization is already present in the candidate pool, does Pendulum select it?* For the headline configuration, the answer is not always. The candidate pool contains a correct formalization for 238 of 258 specifications (92.2%), whereas Pendulum selects a correct formalization for 225 (87.2%). Thus, 13 specifications contain a correct candidate that is available to the selection stage but are nevertheless assigned an incorrect final answer. The resulting 5.0-percentage-point coverage–selection gap is the central empirical observation of the evaluation.

The gap is not confined to the headline model pairing. Holding the Codex candidate pool fixed, changing the assessment model from Gemma-4 31B to Gemma-4 12B leaves coverage unchanged at 92.2% but lowers selection from 87.2% to 85.7%. Changing the synthesis pool affects coverage as well as selection: the Gemma-pool configurations cover 85.2% and 88.6%, with selection rates of 76.7% and 76.0%, respectively. Across all four configurations, a correct candidate is therefore available more often than it is ultimately selected.

The result also holds on the subset of benchmark references left unchanged by the audit, ruling out the possibility that the headline gap is solely a consequence of reference corrections. The corresponding rescored analysis is reported in Appendix B.

4.4 RQ3: Component Effects

RQ2 establishes that candidate availability does not guarantee correct selection. RQ3 asks whether this gap is readily reduced by changing the evidence or mechanisms available to the selection process.

We evaluate twelve controlled interventions spanning stronger selection, richer symbolic evidence, candidate-set construction, input restructuring, and compiler guidance. Each intervention is evaluated with the regression check described above. Full per-intervention results and statistical tests are reported in Appendix C; Table 3 summarizes the main outcome.

The interventions targeting selection, symbolic evidence, candidate-set construction, and input restructuring do not provide consistent evidence that additional machinery closes the gap. Richer traces and other semantic evidence recover some requirements while losing others, and alternative selection procedures similarly trade improvements against regressions. Changing the candidate set can increase candidate availability without producing a corresponding improvement in final selection.

Table 3: Summary of twelve controlled interventions. Changes to selection, symbolic evidence, candidate construction, and input restructuring do not consistently reduce the coverage–selection gap. Compiler guidance improves a separate, mechanically checkable synthesis failure.

Intervention family	Representative changes	Result
Stronger selection	LLM ranking; evidence-based adjudication	no consistent gain
Richer symbolic evidence	traces; paraphrases; operator semantics	no consistent gain
Candidate-set construction	equivalence deduplication; added candidates	no consistent gain
Input restructuring	decomposition; delexicalization	gains offset by regressions
Compiler guidance	specification-language pitfalls	**robust synthesis gain**

Compiler guidance provides a useful contrast. Adding specification-language pitfalls to the synthesis prompt improves that synthesis path by 19 requirements and produces gains of 11, 11, and 18 requirements across three additional synthesis models. This intervention targets deterministic compiler failures rather than interpretation among semantically distinct candidates. In contrast, prescriptive guidance for semantic operator choices does not produce a corresponding improvement and can regress on requirements whose language admits multiple interpretations. The result suggests a boundary for prompt-level intervention: guidance is effective when the target is a mechanically checkable synthesis failure, but does not reliably resolve semantic selection.

4.5 Selection Failure Analysis

RQ1–RQ3 establish that the coverage–selection gap is measurable and persists across the evaluated configurations and interventions. We therefore inspect the residual selection failures to understand what remains after a correct candidate is available. In the headline configuration, 13 determinate specifications contain a correct candidate but receive an incorrect final selection. The correct candidate is ranked second in seven cases and third or fourth in the remaining six. The selected alternatives are generally close formal variants, differing in temporal scope, operator choice, step count, or structural grouping. In several cases, the competing formulas are a bare formula and its globally scoped counterpart; in others, they differ by an additional temporal operator or step.

These cases illustrate the role of symbolic analysis in the pipeline. BLACK can establish whether competing candidates are equivalent and, for non-equivalent candidates, expose the behaviors that distinguish them. The remaining decision is whether those behavioral differences correspond to the interpretation intended by the natural language requirement. The results therefore do not indicate that symbolic analysis is ineffective; rather, the evaluated forms of semantic evidence and selection do not reliably resolve all determinate cases in which multiple formal candidates remain plausible. A per-instance categorization of the selection failures is provided in Appendix B.

5 Discussion

A boundary for verifier-guided selection. Verifier-guided generation is effective when an independent criterion determines whether a candidate satisfies the task: a test suite, proof checker, or other sound validity procedure can establish correctness of the candidate itself (Stechly et al., 2024). NL→LTL selection differs. BLACK can establish equivalence, entailment, consistency, and distinguishing behaviors, but these are relations among formal candidates. The remaining question is whether a candidate faithfully expresses the natural language requirement. When the requirement admits multiple readings, symbolic analysis can expose the difference without determining which reading was intended. The persistence of the coverage–selection gap across the evaluated selection, evidence, and candidate-construction interventions provides empirical evidence for this boundary. This also qualifies the broader LLM-Modulo view (Kambhampati et al., 2024): verification is effective only when the task provides a criterion that the verifier can actually test.

Relation to prior neuro-symbolic systems. Existing neuro-symbolic systems integrate neural generation with symbolic checking, refinement, or certification (Saha et al., 2026; Feng et al., 2025). Pendulum does not replace these architectures or claim superiority on their respective benchmarks. Instead, it makes a distinction that integrated end-to-end evaluation can obscure: candidate generation,

symbolic analysis, and final selection are separate stages with separately measurable outcomes (Figure 2). Reporting candidate coverage alongside top-1 selection reveals whether an observed error arises because the correct formalization was never generated or because it was available but not selected. The latter remains observable even when symbolic analysis is sound.

What the benchmark reveals. The benchmark audit further shows why this distinction matters. Some references required correction, while other specifications did not provide sufficient evidence for a unique formalization and were excluded from the primary evaluation. The remaining selection failures are therefore not explained simply by the presence of incorrect evaluation targets. Across the evaluated cases, the residual errors concentrate on distinctions such as temporal scope and operator choice. The results consequently separate two directions for progress: improving synthesis to increase candidate coverage, and improving requirement specification or selection when multiple semantically distinct candidates remain plausible. The latter is not a problem that solver-backed behavioral analysis alone can settle; it requires information about the intended reading of the requirement.

6 Related Work

Natural language to temporal logic. Natural language-to-temporal-logic translation has been approached through pattern libraries (Dwyer et al., 1999), controlled-syntax compilers such as SALT (Bauer et al., 2006) and FRET (Giannakopoulou et al., 2020), and learned translation and decomposition methods (Chen et al., 2023; Ma et al., 2025; Liu et al., 2023; Cosler et al., 2023; Fuggitti & Chakraborti, 2023). The present work studies a different failure boundary: whether a correct formalization already present among generated candidates is subsequently selected. DS_1^{tricky} (Danso et al., 2026), derived from the Little Tricky Logic benchmark (Greenman et al., 2022), provides the evaluation setting. Concurrently, NeuroSTAR (Saha et al., 2026) combines LTL_f generation with automata-based trace comparison and LLM judging, while NeuroNL2LTL (Quansah & Bonnah, 2026) combines neural generation with satisfiability checking and verifier-guided repair. These systems, along with the approaches summarized in Table 4, integrate generation with symbolic analysis but do not separately measure candidate coverage and final selection.

Neuro-symbolic formalization and verification. Recent systems combine neural generation with symbolic reasoning to constrain, check, or refine model outputs. VeriCoT (Feng et al., 2025) uses SMT-based entailment checking, VIRF (Wu et al., 2026) combines an LLM planner with a deterministic logic tutor, and ARc (An et al., 2026) uses multiple LLM-generated formalizations with semantic cross-checking. AutoSafeLTL (Li et al., 2026) iteratively repairs LTL using language-inclusion checks. These systems illustrate different roles for symbolic reasoning. Pendulum makes generation, analysis, and selection explicit and separates candidate availability from final selection.

Verification and model-based judging. Verifier-guided generation depends on a criterion that can establish whether a candidate satisfies the task. Stechly et al. (2024) examine this distinction for self-critique, while Zheng et al. (2023) document systematic biases in LLM-based judging. Pendulum uses symbolic analysis for formally decidable relationships among candidates; selection additionally requires determining which non-equivalent candidate expresses the intended reading.

Benchmark quality and grounding. Benchmark artifacts and label errors can materially affect evaluation (Gururangan et al., 2018; McCoy et al., 2019; Northcutt et al., 2021). VLTL-Bench (English et al., 2025b) further separates lifting, grounding, translation, and verification in NL→LTL evaluation. Pendulum audits the DS_1^{tricky} references and separately measures candidate coverage and selection. Its alignment scorer (Appendix G.1) additionally evaluates vocabulary correspondence when propositions are not already grounded.

[1]Pattern/compiler and learned translation approaches: Dwyer et al. (1999); Bauer et al. (2006); Giannakopoulou et al. (2020); Chen et al. (2023); Ma et al. (2025); Liu et al. (2023); Cosler et al. (2023); Fuggitti & Chakraborti (2023).

Table 4: Architectural positioning of representative approaches. The rows are not performance comparisons: the systems use different benchmarks, models, and evaluation protocols. 'No" indicates that the approach does not use the corresponding component; 'integrated" indicates that selection or refinement is coupled to symbolic analysis rather than exposed as a separate stage.

System	Neural generation	Symbolic analysis	Selection / refinement	Coverage–selection separation
Pattern/compiler translators[1]	✓	No	No	No
NeuroSTAR (Saha et al., 2026)	✓	✓(DFA traces)	Integrated	No
NeuroNL2LTL (Quansah & Bonnah, 2026)	✓	✓(SAT)	Integrated	No
AutoSafeLTL (Li et al., 2026)	✓	✓(inclusion)	Integrated	No
Pendulum	✓	✓(BLACK)	**Explicit stage**	**Yes**

7 LIMITATIONS

Benchmark scope. Our evaluation uses a single dataset, DS_1^{tricky}, and therefore does not establish that the observed coverage–selection relationship has the same magnitude for other datasets, formal languages, or application domains. The benchmark is also dominated by future-time LTL; although our implementation supports past-time operators, relatively few benchmark instances exercise them.

Benchmark contamination. The benchmark was publicly available before our experiments, and we cannot establish whether the pretrained models used for synthesis or selection had previously encountered its instances. The reported accuracy should therefore not be interpreted as a contamination-free estimate of generalization to unseen requirements. Our primary claim is narrower: under a fixed audited benchmark and solver-backed scoring procedure, candidate coverage and final selection remain empirically distinct quantities.

Ambiguity and adjudication. Determining whether a natural language requirement has a unique intended formalization can require human interpretation. Our audit therefore does not claim that solver-based analysis alone establishes ground truth. We separate specifications for which intent could not be established and exclude them from the primary determinate-set evaluation rather than assigning forced labels. The resulting partition reduces label ambiguity but does not eliminate the subjectivity inherent in interpreting natural language.

Model and intervention dependence. The magnitude of the coverage–selection gap varies across synthesis and judge configurations, and the evaluated interventions do not exhaust the design space. The planted-candidate experiment is likewise supporting evidence rather than a standalone estimate of selector performance. These considerations limit the generality of individual component-level findings, but do not alter the central coverage–selection comparison.

8 CONCLUSION

This paper studied natural language-to-LTL formalization as a separation between candidate generation and candidate selection. On an audited set of 258 determinate specifications, Pendulum contains a semantically correct candidate for 92.2% of cases but selects one correctly for 87.2%. The coverage–selection gap persists across the evaluated changes to candidate generation, symbolic evidence, and selection. These results show that solver-backed analysis can establish relationships among formal candidates without necessarily determining which candidate expresses the intended reading of the natural language requirement. Separating candidate coverage from final selection therefore exposes a failure mode hidden by end-to-end translation accuracy and provides a basis for diagnosing where neuro-symbolic formalization systems succeed and fail.

AI USE STATEMENT

Generative AI tools were used in supporting roles during system development, benchmark auditing, literature search, and manuscript preparation. During system development, AI tools assisted with implementation of synthesis modules, the evaluation harness, and supporting tooling. During the benchmark audit, AI-assisted screening was used to identify potentially problematic references for author review; model output was not used as ground truth. Each accepted correction was independently adjudicated by the authors against the natural language requirement and checked using solver-backed semantics.

Generative AI tools were also used for literature search and for typographical, grammatical, and language-level corrections to the manuscript. All AI-assisted changes were reviewed by the authors. The use of generative AI did not replace author decisions about the research questions, experimental methodology, benchmark adjudication, or interpretation of the results. No generative AI model was trained or fine-tuned for this work.

The specific AI systems, models, and versions used in these roles are documented in the accompanying artifact and reproducibility materials. No model is an author. The authors take responsibility for the final content of this work, including its claims, code, data, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The benchmark contains no personal data, identifiable individuals, or offensive material. It consists of synthetic and paraphrased software-requirement sentences paired with temporal-logic formalizations. The benchmark corrections introduce no additional sensitive content. Natural language-to-temporal-logic translation can support verification, monitoring, and controller synthesis for software and cyber-physical systems, including safety-critical applications. The results therefore highlight the risk of treating model-generated formalizations as self-validating when the underlying natural language requirement is underspecified.

REPRODUCIBILITY STATEMENT

We release the systems, corrected benchmark, alignment scorer, and evaluation artifacts needed to reproduce the reported experiments. The anonymous artifact repository is Anonymous (2026). The repository documents the symbolic substrate, fixed pipeline, unconstrained agent, model configurations, inference backends, and local versus API-based execution. It also provides the benchmark audit artifacts and correction rationales, per-row scoring data, and analysis scripts needed to recompute the reported results against both the audited and original references. Determinism considerations and the procedure for reproducing the all-local experiments are documented with the released artifacts.

REFERENCES

Chenyang An, Sam Bayless, Stefano Buliani, Darion Cassel, Byron Cook, Duncan Clough, Rémi Delmas, Nafi Diallo, Ferhat Erata, Nick Feng, Dimitra Giannakopoulou, Aman Goel, Aditya Gokhale, Joe Hendrix, Victor Heorhiadi, Marc Hudak, Dejan Jovanović, Andrew M. Kent, Benjamin Kiesl-Reiter, Jeffrey J. Kuna, Nadia Labai, Joseph Lilien, Divya Raghunathan, Zvonimir Rakamarić, Niloofar Razavi, Michael Tautschnig, Ali Torkamani, Nathaniel Weir, Michael W. Whalen, and Jianan Yao. A neurosymbolic approach to natural language formalization and verification. In *Computer Aided Verification: 38th International Conference, CAV 2026, Lisbon, Portugal, July 26–29, 2026, Proceedings, Part II*, pp. 601–617, Berlin, Heidelberg, 2026. Springer-Verlag. ISBN 978-3-032-32525-9. doi: 10.1007/978-3-032-32526-6_28. URL https://doi.org/10.1007/978-3-032-32526-6_28.

Anonymous. Anonymous artifact. https://anonymous.4open.science/status/pendulum-2D7F, 2026.

Andreas Bauer, Martin Leucker, and Jonathan Streit. SALT–-structured assertion language for temporal logic. In *Proceedings of the Eighth International Conference on Formal Engineering Methods*, volume 4260 of *Lecture Notes in Computer Science*, 2006.

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. NL2TL: transforming natural languages to temporal logics using large language models. *CoRR*, abs/2305.07766, 2023. doi: 10.48550/ARXIV.2305.07766. URL https://doi.org/10.48550/arXiv.2305.07766.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part II*, pp. 383–396, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-37702-0. doi: 10.1007/978-3-031-37703-7_18. URL https://doi.org/10.1007/978-3-031-37703-7_18.

Priscilla Kyei Danso, Mohammad Saqib Hasan, Niranjan Balasubramanian, and Omar Chowdhury. Syntax is easy, semantics is hard: Evaluating llms for ltl translation. In *Proceedings of the 2026 ACM Secure Development Conference*, SecDev '26, pp. 152–168, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400726026. doi: 10.1145/3805773.3806005. URL https://doi.org/10.1145/3805773.3806005.

Alexandre Duret-Lutz, Ala Eddine Hassen, Natalie Lewkowicz, Antoine Fauchille, Thibaud Michaud, Eryk Renault, and Xu Xu. Spot 2.0 — a framework for ltl and ω-automata manipulation. In *Automated Technology for Verification and Analysis (ATVA 2016)*, volume 9938 of *Lecture Notes in Computer Science*, pp. 122–129. Springer, 2016. doi: 10.1007/978-3-319-46520-3_8.

Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pp. 411–420, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130740. doi: 10.1145/302405.302672. URL https://doi.org/10.1145/302405.302672.

William English, Chase Walker, Dominic Simon, Sumit Jha, and Rickard Ewetz. Verifiable natural language to linear temporal logic translation: A benchmark dataset and evaluation suite. *arXiv preprint arXiv:2507.00877*, 2025a.

William H English, Chase Walker, Dominic Simon, Sumit Kumar Jha, and Rickard Ewetz. Verifiable natural language to linear temporal logic translation: A benchmark dataset and evaluation suite, 2025b. URL https://arxiv.org/abs/2507.00877.

Yu Feng, Nathaniel Weir, Kaj Bostrom, Sam Bayless, Darion Cassel, Sapana Chaudhary, Benjamin Kiesl-Reiter, and Huzefa Rangwala. VeriCoT: Neuro-symbolic chain-of-thought validation via logical consistency checks, 2025. URL https://arxiv.org/abs/2511.04662. ICLR 2026 Poster.

Francesco Fuggitti and Tathagata Chakraborti. Nl2ltl - a python package for converting natural language (nl) instructions to linear temporal logic (ltl) formulas. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i13.27068. URL https://doi.org/10.1609/aaai.v37i13.27068.

Luca Geatti, Nicola Gigante, and Angelo Montanari. BLACK: A fast, flexible and reliable LTL satisfiability checker. In *Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY)*, volume 2987 of *CEUR Workshop Proceedings*, pp. 7–12, 2021. URL https://ceur-ws.org/Vol-2987/paper2.pdf.

Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. Generation of formal requirements from structured natural language. In *Requirements Engineering: Foundation for Software Quality: 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24–27, 2020, Proceedings*, pp. 19–35, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-44428-0. doi: 10.1007/978-3-030-44429-7_2. URL https://doi.org/10.1007/978-3-030-44429-7_2.

Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Little tricky logic: Misconceptions in the understanding of ltl. *The Art, Science, and Engineering of Programming*, 7

(2), October 2022. ISSN 2473-7321. doi: 10.22152/programming-journal.org/2023/7/7. URL http://dx.doi.org/10.22152/programming-journal.org/2023/7/7.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 107–112, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2017. URL https://aclanthology.org/N18-2017/.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: Llms can't plan, but can help planning in llm-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Junle Li, Siqi Chen, Jiakai Li, Meiqi Tian, and Bingzhuo Zhong. Automatic generation of safety-compliant linear temporal logic via large language model: A self-supervised framework, 2026. URL https://arxiv.org/abs/2503.15840.

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 Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1084–1110. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/liu23d.html.

Zhi Ma, Cheng Wen, Zhexin Su, Xiao Liang, Cong Tian, Shengchao Qin, and Mengfei Yang. Bridging natural language and formal specification–automated translation of software requirements to ltl via hierarchical semantics decomposition using llms. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1208–1220. IEEE Press, 2025. doi: 10.1109/ASE63991.2025.00104. URL https://doi.org/10.1109/ASE63991.2025.00104.

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

Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=XccDXrDNLek.

Paulo Pirozelli, Marcos M. José, Paulo de Tarso P. Filho, Anarosa A. F. Brandão, and Fabio G. Cozman. Assessing logical reasoning capabilities of encoder-only transformer models. In *Neural-Symbolic Learning and Reasoning: 18th International Conference, NeSy 2024, Barcelona, Spain, September 9–12, 2024, Proceedings, Part I*, pp. 29–46, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-71166-4. doi: 10.1007/978-3-031-71167-1_2. URL https://doi.org/10.1007/978-3-031-71167-1_2.

Paapa Kwesi Quansah and Ernest Bonnah. Neuronl2ltl: A neurosymbolic framework for natural language translation of linear temporal logic. In *Formal Techniques for Distributed Objects, Components, and Systems: 46th IFIP WG 6.1 International Conference, FORTE 2026, Held as Part of the 21st International Federated Conference on Distributed Computing Techniques, DisCoTec 2026, Urbino, Italy, June 8–12, 2026, Proceedings*, pp. 56–73, Berlin, Heidelberg, 2026. Springer-Verlag. ISBN 978-3-032-28186-9. doi: 10.1007/978-3-032-28187-6_4. URL https://doi.org/10.1007/978-3-032-28187-6_4.

Joy Saha, Trey Woodlief, Sebastian Elbaum, and Matthew B. Dwyer. Neurostar: Automata-guided neuro-symbolic specification formalization, 2026. URL https://arxiv.org/abs/2609.03161.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks, 2024. URL https://arxiv.org/abs/2402.08115.

Jun Wang, David Smith Sundarsingh, Jyotirmoy V. Deshmukh, and Yiannis Kantaros. Conformalnl2ltl: Translating natural language instructions into temporal logic formulas with conformal correctness guarantees. *CoRR*, abs/2504.21022, 2025. doi: 10.48550/ARXIV.2504.21022. URL https://doi.org/10.48550/arXiv.2504.21022.

Feiyu Wu, Xu Zheng, Yue Qu, Zhuocheng Wang, Zicheng Feng, and HUI LI. Grounding generative planners in verifiable logic: A hybrid architecture for trustworthy embodied ai. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 39368–39420, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/423295ba2cea13cda44dc0a7d48ae244-Paper-Conference.pdf.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

A Experimental Configurations

This appendix specifies the model assignments, inference backends, and execution conditions used to produce the reported results. Unless otherwise stated, configurations use the fixed Pendulum control flow described in §2.

Model seats. Pendulum assigns named model seats to AP extraction, orchestration and finalization, the two synthesis paths, optional pattern lookup, selection judging, optional deterministic verification, and the one-shot baseline. Each seat can be independently overridden through an environment variable. When a seat is not explicitly configured, it resolves to the default local ≈30B model. This makes the all-local configuration the default rather than silently mixing model providers. The unconstrained agent uses a separate orchestrator seat and therefore does not inherit the pipeline's model assignment.

Inference backends. The headline configuration uses a proprietary coding model through its official command-line client for the synthesis seat. The client runs in non-interactive, sandboxed mode; authentication and billing are handled by the client rather than an HTTP API. The local configurations use the available local inference backends. The command-line clients do not expose token-level log probabilities, and their temperature controls are ignored; the corresponding optional confidence-based behavior is therefore disabled when those signals are unavailable. Transient client errors and rate limits are retried with backoff before an invocation is classified as an infrastructure failure. Infrastructure failures are excluded from the evaluation denominator rather than counted as incorrect predictions.

Determinism. The `gguf` inference engine honors the configured decoding seed and produces reproducible temperature-zero outputs. The MLX backend used for the default local model does not honor the seed and can produce different temperature-zero outputs across identical runs. This distinction affects small intervention deltas: an apparent one- or two-requirement change on an MLX-backed configuration should not be interpreted as a reliable treatment effect without replication. We therefore treat such single-run differences as nondeterministic variation unless replicated across runs or reproduced with the seed-reproducible `gguf` backend. The intervention reported in Table 11 provides an example: the apparent gain did not reproduce under the reproducible backend.

Hardware. All local-model experiments were executed on a single workstation with approximately 80 GB of unified memory, sufficient to host the default ≈30B model at the evaluated precision. We did not evaluate a smaller-model hardware configuration as a separate experimental condition.

Reproducing the reported configurations. The all-local configuration can be reproduced without proprietary API access by leaving the model seats at their local defaults and selecting the local inference backend. The headline configuration additionally requires an authenticated installation of

Table 5: Manual categorization of the 13 headline selection failures: cases in which a semantically correct candidate was present in the candidate pool but an incorrect candidate was ultimately selected. 238 any-of − 225 top-1 = 13 on the 258-reference determinate subset, matching Table 2.

Failure type	Count
Global-scope difference (bare vs. G-wrapped)	7
Temporal operator or step-count difference	5
Structural / associativity difference	1
Total	**13**

the corresponding command-line client. The released artifact contains the experiment drivers, model-seat assignments, configuration files, and commands needed to reproduce the reported experiments.

B Additional Results

This appendix provides the detailed results supporting the evaluation in §4. The main text reports the audited 258-specification evaluation and the corresponding coverage–selection gap. The results below provide additional per-configuration measurements, statistical tests, selection-failure analysis, and historical comparisons. Unless explicitly marked as a historical anchor, results use the audited correctness targets described in §3.

B.1 Selection Failures

The headline configuration contains a correct candidate for 238 of the 258 determinate specifications, but selects a correct candidate for 225. The remaining 13 cases therefore isolate the selection failure: the correct formalization is available to the selection stage, but another candidate is committed.

Table 5 summarizes these 13 cases by the difference between the selected formula and the audited reference. The correct candidate is ranked second in seven cases and third or fourth in the remaining six. Thus, the failures occur after synthesis has already supplied a correct formalization; they are not failures of candidate coverage.

The largest category consists of scope differences between otherwise closely related candidates. In seven cases, the selected formula and the correct candidate differ only in whether a global `G` scope is applied; several of these are rows where only two candidates survived filtering in the first place, so the selector chose between exactly two formal readings of a single synthesized formula and picked the wrong one. The other six cases involve a different temporal operator, step count, or structural grouping. These cases are consistent with the distinction evaluated in the main text: BLACK can establish the formal relationship between two candidates, but the selection stage must still determine which candidate matches the natural language requirement.

B.2 Robustness to Benchmark Corrections

A potential concern is that the observed coverage–selection gap could result from correcting the benchmark references. We therefore separately evaluate references that were changed during the audit and references that were left unchanged.

The gap remains present in both partitions across the rescored configurations. The magnitude of the difference also varies across configurations rather than systematically favoring either the corrected or unchanged references. Thus, the primary observation is not dependent on the subset of references that required correction.

The corrected and unchanged partitions should therefore be interpreted as a robustness check rather than as separate benchmark evaluations. The audited 258-specification set remains the primary evaluation target.

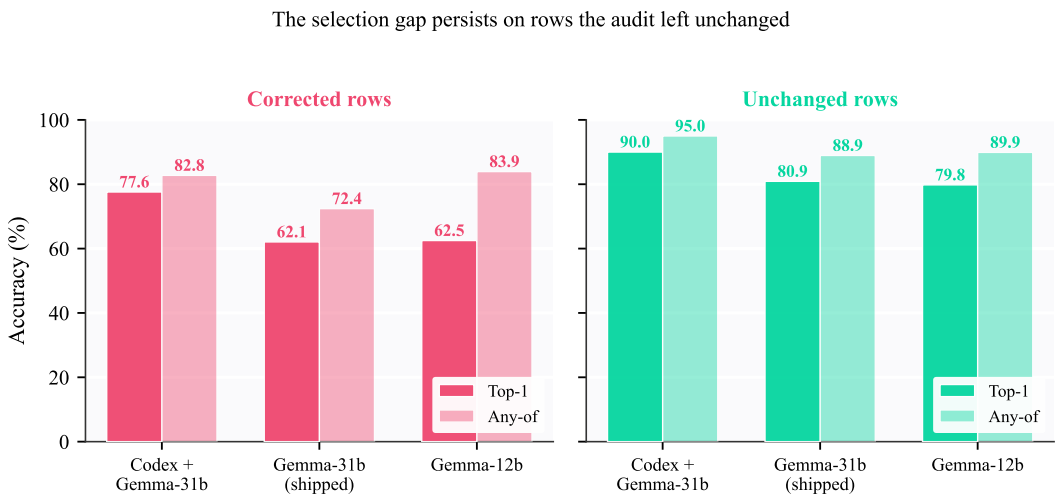


Figure 3: Coverage–selection gap on benchmark references corrected during the audit and references left unchanged. The gap remains observable in both partitions across the evaluated configurations.

Table 6: Historical 303-instance results obtained before the benchmark audit, scored against the then-current, pre-audit references. These results are retained as a comparison point for experiments conducted before the audited evaluation and are not the primary results of this paper.

System	Top-1	Any-of
Local 30B, one-shot	164/303	—
Proprietary, one-shot	182/303	182/303
Proprietary, 5-candidate list	174/303	210/303
Fixed pipeline, all-local	175/303	196/303
Fixed pipeline, historical headline	197/303	215/303

B.3 Historical 303-Instance Anchor

The original parseable benchmark contains 303 instances (306 minus three rows excluded only for an unrelated pre-audit syntax repair) and is retained as a historical anchor for comparisons with experiments conducted before the benchmark audit described in §3. These results are not used for the primary RQ1–RQ3 measurements.

Before the audit, the headline staged pipeline obtained 197/303 top-1 selections (65.0%) and contained a correct candidate for 215/303 specifications (70.9%) under the then-current, pre-audit references. These values are reported only to preserve the historical comparison with earlier one-shot and candidate-list experiments. They should not be compared numerically with the audited 258-instance headline result, because the correctness targets and evaluation population differ.

The two multi-candidate systems in this comparison – the historical pipeline and the proprietary 5-candidate list – have similar measured candidate-pool precision (0.374 and 0.380 respectively, both BLACK-scored against the then-current reference), while their downstream top-1 selection differs (0.650 vs. 0.571). Both figures are far below the single-formula baseline's trivial precision of 0.597 (one candidate, so pool precision equals top-1). This is the quantitative basis for treating the pipeline's historical advantage over the same model's own ranked list as a difference in selection rather than in candidate-pool quality.

For reference, rescoring the same 303-row population against the corrected ground truth – rather than the then-current, pre-audit references, but still including the 48 specifications later classified as ambiguous – gives 242/303 (79.9%) top-1 and 261/303 (86.1%) any-of for the headline configuration. This is consistent with, though not directly comparable to, the audited 258-specification result in the main text, since the 303-row population still includes the ambiguous specifications excluded from the primary evaluation.

Table 7: Rank-graded measurements for the four synthesis-pool / judge configurations of Table 2, on the 303-row historical anchor with the then-current, pre-audit references.

synth. pool	judge	top-1	any-of	lin.	MRR	NDCG
codex py+salt	gemma-31B	**197**	**215**	**.690**	**.674**	**.682**
codex py+salt	gemma-12B	189	215	.680	.656	.669
gemma py+salt	gemma-31B	175	196	.626	.607	.617
gemma py+salt	gemma-12B	173	198	.638	.614	.627

Table 8: Wilson 95% CIs behind the historical 303-instance comparison of Table 6, computed on the then-current, pre-audit references.

system	top-1 [95% CI]	any-of [95% CI]	MRR / NDCG
codex 1-shot	.597 [.541,.651]	.597 [.541,.651]	.597 / .597
codex 5-cand. list	.571 [.515,.625]	.690 [.636,.739]	.624 / .641
pipeline, historical headline	**.650** [.592,.699]	**.710** [.653,.755]	**.674 / .682**

B.4 Configuration Sensitivity

The main text reports four combinations of synthesis pool and assessment model (Table 2). Table 7 gives the corresponding rank-graded measurements (linear, MRR, NDCG) for the same four configurations on the 303-row historical anchor with pre-audit references, retained for comparison with earlier reporting conventions on this benchmark.

Changing the assessment model while holding the Codex synthesis pool fixed leaves candidate coverage unchanged but changes final selection. Changing the synthesis pool changes both coverage and selection. The distinction between the two quantities therefore remains visible when either side of the pipeline is varied.

B.5 Paired Statistical Tests

Because the same benchmark specifications are evaluated by each compared system, the primary binary comparisons use exact paired McNemar tests. Confidence intervals for proportions use Wilson intervals. Rank-sensitive metrics are compared using paired bootstrap resampling with 10,000 resamples and seed 7.

For the historical 303-instance comparison, the staged pipeline differs significantly from the corresponding one-shot and five-candidate-list baselines in top-1 selection. The exact paired comparisons are:

- pipeline vs. one-shot: $b = 19$, $c = 4$, $p = 0.0026$;
- pipeline vs. five-candidate list: $b = 29$, $c = 6$, $p = 0.00012$;
- five-candidate list vs. one-shot: $b = 11$, $c = 19$, $p = 0.20$.

For candidate coverage, the pipeline and five-candidate list are not significantly different in this historical comparison ($b = 26$, $c = 21$, $p = 0.56$), whereas both exceed the corresponding one-shot coverage measurement ($b = 36, c = 3, p = 3.6 \times 10^{-8}$ and $b = 35, c = 7, p = 1.5 \times 10^{-5}$ respectively). Rank-graded, paired-bootstrap differences (pipeline minus codex 5-candidate list) are $+0.032\,[-0.007, +0.071]$ (linear, not significant), $+0.048\,[+0.013, +0.083]$ (MRR), and $+0.040\,[+0.004, +0.076]$ (NDCG).

These tests support the historical comparison only. The audited 258-instance headline result in the main text is interpreted primarily through the coverage–selection decomposition rather than as a leaderboard comparison against the historical systems.

B.6 Selection and Evidence Ablations

Several experiments vary the evidence available to selection while keeping the candidate set fixed. These experiments test whether additional semantic evidence consistently improves the final decision.

Increasing the assessment model from Gemma-4 12B to Gemma-4 31B improves top-1 selection on the Codex candidate pool from 85.7% to 87.2% ($b = 8$, $c = 0$, exact McNemar $p = 0.0078$). On the weaker Gemma candidate pool, the corresponding comparison is not significant ($b = 5$, $c = 3$, $p = 0.73$).

Changing deterministic finalization to evidence-based LLM adjudication also produces no reliable improvement in the controlled comparison ($b = 2$, $c = 1$, $p = 1.0$). Adding retrieval-based SALT support does not produce a significant change in synthesis output ($b = 31$, $c = 28$, $p = 0.79$).

The broader intervention suite (Appendix C) shows the same pattern. Changes to candidate construction, semantic evidence, and selection can recover individual requirements, but the gains are accompanied by regressions elsewhere. Consequently, these interventions do not provide consistent evidence that the coverage–selection gap can be removed by adding more evidence or a different selection mechanism.

B.7 Candidate Pool Effects

Candidate-set size and composition affect coverage independently of selection. Adding a second model's one-shot translation as an additional candidate increases candidate coverage by as many as 13 requirements in the evaluated setting (the largest single coverage addition measured in the study). However, the corresponding top-1 result decreases by between one and three requirements across the tested sources; a deliberately planted, correct extra candidate is ranked first by the judge only 3 of 14 times (21%). Pooling grounding, decomposition, and FRET candidates onto the two-path pool shows the same pattern at a different scale: candidate coverage rises while top-1 selection falls (pre-audit, 303-row anchor: any-of $184 \rightarrow 193$, top-1 $171 \rightarrow 166$).

This experiment provides a direct example of why coverage and selection should be reported separately. Additional synthesis can place more correct formalizations in the candidate pool without guaranteeing that the final selector will choose them.

A judge-free agreement selector provides a complementary comparison. On the audited 258-specification subset, the full pooled candidate set contains a correct candidate for 250 specifications (96.9%), while ranking that pool purely by cross-source agreement – how many independently-produced, BLACK-deduplicated candidates land on the same formula, with no LLM judgment in the loop – selects a correct candidate for 225 specifications (87.2%), matching the contrastive-judge pipeline's own top-1 selection while avoiding its candidate-crowding sensitivity. Thus, a selector that uses cross-source agreement rather than an LLM judge does not convert the full candidate-pool coverage into equivalent top-1 selection either; recovering the residual gap is not simply a matter of which selection mechanism is used (Appendix E).

B.8 Vocabulary Grounding

Providing the atomic propositions explicitly improves performance for both the fixed pipeline and the unconstrained agent. On the historical 303-instance anchor, alignment-scored, removing the supplied vocabulary decreases top-1 selection by 15.8 percentage points for the historical pipeline headline (.650→.492), by 10.9 points for the all-local pipeline (.579→.470), and by 12.6 points for the corresponding gemma-orchestrated agent configuration (.593→.467).

The effect is therefore not specific to the selection mechanism. Both systems lose double-digit percentage points when proposition grounding is removed, indicating that vocabulary specification is an independent source of difficulty in natural language-to-LTL translation, separate from the coverage–selection distinction this paper studies. On DS_1^{tricky} the tax is comparatively small because the atoms are near-trivial; on grounding-stress data a comparable pipeline drops from ≈89% lifted translation to ≈30% end-to-end (English et al., 2025b). Every published number on this benchmark, ours included, measures translation given a vocabulary.

Table 9: Orchestration comparison, historical 303-instance anchor, both vocabulary conditions. No-AP rows are AP-aligned (Appendix G.1) and re-paired per comparison, excluding a row where either arm failed.

system, gold APs	top-1	any-of
pipeline, historical headline (codex + 31B judge)	**197 (.650)**	**215 (.710)**
agent, codex orchestrator (local coders)	186 (.614)	197 (.650)
agent, gemma-31B orchestrator (all-local)	179 (.593)	184 (.609)
pipeline, all-local	175 (.579)	196 (.649)
system, no APs (AP-aligned)	**top-1**	**any-of**
agent, claude-opus orchestrator	**170 (.563)**	**187 (.619)**
agent, codex orchestrator	166 (.550)	186 (.616)
pipeline, historical headline (codex coders)	148 (.492)	170 (.565)
one-shot local 31B (no scaffolding)	142 (.470)	142 (.470)
agent, gemma-31B orchestrator	141 (.467)	149 (.493)

B.9 Orchestration Comparison

The unconstrained agent (Appendix F.3) provides a comparison with a system in which an LLM controls tool use rather than following the fixed Pendulum control flow. Table 9 gives the full comparison on the historical 303-instance anchor, both vocabulary conditions.

With gold atomic propositions supplied, the gemma-orchestrated agent and the all-local fixed pipeline are not significantly different (179 vs. 175, $p = 0.52$), and the codex-orchestrated agent is not significantly different from the gemma-orchestrated agent (186 vs. 179, $p = 0.47$). When atomic propositions are not supplied, the difference between orchestration strategies becomes larger: the claude-orchestrated agent differs significantly from the historical pipeline headline (170 vs. 148, $p = 0.0062$), as does the codex-orchestrated agent (166 vs. 148, $p = 0.0096$), while the two frontier orchestrators do not differ significantly from each other (170 vs. 166, $p = 0.65$). Inspecting the agent's trajectories on the development split, it calls two of its nine tools – one synthesis call and one paraphrase check – and does not invoke the equivalence, entailment, lattice, or trace tools that the fixed pipeline's selection stage is built around; most of its committed answers are verbatim a synthesis module's output. The same ordering (agent and pipeline not significantly different at equal model and vocabulary; a frontier orchestrator differing from the pipeline without gold APs) reproduces on a 226-row held-out split never used for tuning, and on a second, non-templated corpus of real requirement English restricted to its 99 temporal-logic rows.

The orchestration experiments do not establish that one orchestration strategy is generally superior. They show that the measured effect of orchestration depends on whether the system is given the proposition vocabulary, providing an additional reason to treat grounding and selection as separate factors.

Audited companion figures. Table 9 reports the historical 303-instance anchor for continuity with earlier reporting. For the subset of agent configurations with a stored, unmodified candidate record, Table 10 gives the same comparison rescored against the audited 258-specification subset, using the alignment procedure of Appendix G.1 for the no-AP rows. This is a deterministic re-verification of already-generated candidates against the corrected reference; no additional model call was made. The pipeline rows are omitted here because the corresponding all-local and historical-headline pipeline runs were rescored on the 258-specification subset directly in §4 rather than through this agent-comparison harness.

Every configuration improves substantially under the corrected reference, consistent with the same correction effect reported for the pipeline headline in §4. The ordering among agents is materially similar to the historical comparison (codex orchestrator ahead of the gemma-orchestrated agent in both vocabulary conditions), which is evidence against the corrected reference selectively favoring one configuration over another.

Prompt-level exploration. A more permissive orchestrator prompt was also evaluated, explicitly encouraging the agent to consult its second synthesis path before finalizing. Relative to the default

Table 10: The subset of Table 9's agent configurations rescored against the audited 258-specification subset. No-AP rows are aligned per-candidate (Appendix G.1); this differs from Table 9's no-AP rows, which align once per row and can therefore understate correctness when a row's candidates do not all share the same atomic-proposition vocabulary. Denominators of 257 reflect one row without a terminal outcome.

system, gold APs	top-1	any-of
agent, codex orchestrator (local coders)	213/258 (.826)	221/258 (.857)
agent, gemma-31B orchestrator (all-local)	201/257 (.782)	204/257 (.794)
system, no APs (AP-aligned)	**top-1**	**any-of**
agent, claude-opus orchestrator	206/258 (.798)	218/258 (.845)
agent, codex orchestrator	197/257 (.767)	210/257 (.817)
agent, gemma-31B orchestrator	166/257 (.646)	170/257 (.661)

Table 11: Controlled interventions evaluated across candidate synthesis, semantic evidence, selection, input representation, and candidate-set construction. Effects are measured with paired regression checks where applicable. The experiments do not isolate a single common mechanism; they test whether plausible changes to the pipeline consistently reduce the coverage–selection gap.

Pipeline change	Observed effect	Role
Specification-language pitfalls prompt	$+19$ pooled; $+11/+11/+18$ across three models	Synthesis
Executable operator semantics	$0/+2$ across the audited models; $-1/-1$ on the 303-instance anchor	Synthesis
Proprietary vs. local judge	No consistent advantage after correcting the disagreement set	Selection
LLM vs. deterministic ranking	-33 on 258; -37 on 303	Selection
Evidence-based adjudicator	0 on 258; $+1$ on 303	Selection
Per-paraphrase audit judge	-1 on 258; -1 on 303	Selection
Trace explanation	-1 on 258; -2 on 303	Evidence
Per-leaf synthesis	-2 on hard rows; -1 on solved rows	Synthesis
Delexicalization	-3 on hard rows; -4 on solved rows	Input
Requirements-formalizer path	No uniquely solved requirements after audit	Synthesis
Specification-manual retrieval	Approximately no end-to-end change	Retrieval
Logical-equivalence deduplication	-2 on 258 affected rows; -3 on 303 affected rows	Candidates

prompt, this configuration increased SALT usage from 5% to 69% of runs, the average candidate count from 1.16 to 1.54, and the average call count from 2.23 to 3.11, but produced no measurable change in accuracy (top-1 47 vs. 47, any-of 48 vs. 49, both $p = 1.000$ on the development split) at 64% higher compute cost. The default prompt is used throughout the reported results.

C Controlled Interventions

The interventions in this appendix test whether the coverage–selection gap changes under controlled modifications to candidate synthesis, semantic evidence, selection, input representation, and candidate-set construction. Each intervention corresponds to one experiment in the released `baseline_experiments/` suite. Unless otherwise stated, effects are reported using a paired regression check: the number of requirements improved by the intervention minus the number regressed by it, evaluated on a common set of requirements.

The interventions are heterogeneous by design. They do not constitute a single ablation of one component. Instead, they test whether the observed coverage–selection gap is sensitive to several plausible changes to the pipeline. The primary evaluation remains the audited 258-specification subset; the 303-instance results are retained where they provide a historical comparison with experiments conducted before the audit. Across these twelve changes to synthesis, semantic evidence, selection, and candidate construction, no evaluated intervention consistently eliminated the coverage–selection gap.

1. Specification-language pitfalls. The SALT synthesis path was given a compiler-corrected prompt containing a known syntax error and five specification-language pitfalls identified from failed compilations. The change improves mechanical compilation success by 19 requirements on the pooled evaluation and by 11, 11, and 18 requirements across three additional synthesis models. Because the intervention targets a deterministic interface between generated specifications and the compiler, the effect is unchanged by the benchmark audit. This provides a control case: prompt guidance produces a reproducible improvement when the failure criterion is mechanically checkable.

2. Executable operator semantics. The code-completion synthesis path was additionally given executable descriptions of temporal-operator semantics. Across four evaluated models, the original measurement was flat to negative. On the audited subset, the two models with both arms recorded show net effects of 0 for Gemma-12B and $+2$ for Gemma-31B; on the 303-instance anchor both comparisons are -1. The results do not show a consistent improvement from replacing existing semantic descriptions with longer executable descriptions. The additional representation also increases the prompt by approximately 2,300 tokens.

3. Judge model. The judge model was varied while holding the candidate set fixed. The original comparison suggested differences among local and proprietary judges, but the relevant disagreement set changes after the benchmark audit: it contains 38 specifications on the 258-reference subset and 44 on the 303-instance anchor. On these rescored sets, Gemma-12B, Codex, and Claude obtain 25/38, 22/38, and 22/38 selections on the 258-reference subset, and 27/44, 26/44, and 25/44 on the 303-instance anchor, respectively.

These measurements do not show a consistent advantage for a proprietary judge. They also illustrate why the pre-audit disagreement-row analysis should not be interpreted independently of the audited references: some apparent synthesis disagreements were attributable to incorrect benchmark references.

4. LLM versus deterministic ranking. The final ranking mechanism was varied while keeping the verified candidate set fixed. Replacing the deterministic $(\text{SUPPORTS} - \text{REFUTES})$ ranking with an LLM orchestrator produced a net decrease of 33 requirements on the audited subset and 37 on the 303-instance anchor when errored rows are counted as misses, following the original evaluation procedure. The LLM-based ranking also incurred additional execution failures. The deterministic rule is therefore used in the reported pipeline. This comparison concerns the evaluated ranking mechanisms; it does not establish that deterministic ranking is generally preferable to LLM-based selection.

5. Evidence-based adjudication. A tool-calling adjudicator was evaluated on a fixed Codex candidate pool. The adjudicator had access to paraphrases, the entailment relationships, and distinguishing traces rather than receiving a single flat evidence record. Relative to the comparison selector, the paired regression is 0 on the 258-reference subset (one requirement fixed and one broken) and $+1$ on the 303-instance anchor (two fixed and one broken). The change therefore affects only a small number of requirements and does not provide evidence of a systematic reduction in the selection gap.

6. Per-paraphrase audit evidence. A second selection intervention required the judge to examine candidate paraphrases individually against a deterministic literal reading. The initial MLX run showed a small positive effect, but the MLX backend does not honor the configured decoding seed. Repeated evaluation with the seed-reproducible `gguf` backend produced identical base and audit results.

The intervention was subsequently rescored with the Gemma-31B judge used for the headline configuration. On the audited 258-reference subset, the net effects for the TNL, TNL-only, distinguishing-trace, and audit arms were $+2$, 0, 0, and -1, respectively. On the 303-instance anchor they were $+1$, -2, 0, and -1. Thus, the original positive observation is not reproduced under controlled decoding, and the additional audit evidence does not produce a consistent improvement.

7. Trace explanations. The judge was also given deterministic English explanations describing why a candidate violates a solver-generated forbidden trace. Four evidence conditions were compared: the base evidence, forbidden-trace explanations, distinguishing traces, and the combination of traces and explanations.

On the audited subset, the corresponding paired effects are -1, 0, and -2 for the three augmented conditions. On the 303-instance anchor they are -2, 0, and -2. The results remain close to the base condition and do not indicate that converting solver traces into natural language explanations consistently improves selection.

8. Per-leaf synthesis. Compound requirements were decomposed into individual leaves, synthesized separately, and then recombined. The intervention was evaluated on previously hard and previously solved requirements. On the audited benchmark, the paired effect is -2 on the hard subset and -1 on the solved subset, with the same direction on the 303-instance anchor.

The decomposition therefore does not recover additional correct formalizations reliably enough to offset regressions on requirements that were already solved. This experiment does not establish that decomposition is ineffective in general; it shows that the evaluated decomposition procedure does not provide a consistent improvement for this benchmark.

9. Delexicalization. The input sentence was transformed by replacing proposition-referring phrases with atomic-proposition symbols while retaining temporal and logical language. On the audited benchmark, the paired effect is -3 on the real-world hard subset and -4 on the previously solved subset. The corresponding effects on the 303-instance anchor are -2 and -3.

The additional preprocessing therefore introduces regressions that offset the intended reduction in lexical variation. The result does not support delexicalization as a consistent improvement to the evaluated synthesis path.

10. Requirements-formalizer path. A FRET-based controlled-English formalization path was evaluated both as an independent synthesis source and as an additional candidate source. Its standalone accuracy is approximately 17–19% for the evaluated FRET models, compared with substantially higher accuracy for the two primary synthesis paths. Substituting the path into the pipeline also regresses on requirements already solved by the existing synthesis procedures.

The original evaluation identified two requirements uniquely solved by this path. After rescoring against the audited references and the full comparison set, no requirement remains uniquely solved by FRET. The two previously unique cases remain individually correct under the corrected references, but are also solved by another evaluated synthesis source. Thus, the audited results do not provide evidence that this path adds unique candidate coverage.

11. Specification-manual retrieval. The SALT authoring path uses retrieval over the specification manual. Manual probes showed that the original retrieval configuration failed to retrieve relevant material for a substantial fraction of tested queries. Two repaired retrieval configurations corrected these retrieval failures, but neither produced a measurable end-to-end change in the evaluated output.

This experiment separates retrieval quality from downstream synthesis quality: the retrieval component can be improved mechanically without producing a corresponding improvement in final formalization accuracy. Because the intervention concerns retrieval behavior rather than benchmark labels, its measurement is unchanged by the benchmark audit.

12. Logical-equivalence deduplication. The candidate pool was deduplicated using BLACK-checked logical equivalence rather than canonical surface form. The comparison was restricted to rows where the two deduplication procedures could produce different candidate sets: 61 rows on the audited subset and 81 on the 303-instance anchor.

The paired effect is -2 on the audited affected population and -3 on the 303-instance population. Thus, removing logically equivalent candidates does not improve the measured selection outcome in the evaluated setting. The intervention also requires additional pairwise solver checks as the candidate set grows. The result suggests that the residual selection failures are not explained simply by duplicate surface representations of the same logical formula.

Table 12: Defect classes among the 55 confirmed reference corrections.

Defect class	Count	Example
Wrong scope	36	n032
Wrong temporal operator	8	n072
Structural / other	4	n014
Wrong negation	3	n056
Wrong counting	2	n133
Dropped condition	1	n033
Other variable reference	1	n199
Total	**55**	

D BENCHMARK AUDIT: CORRECTIONS AND AMBIGUITY PARTITION

This appendix documents the procedure used to audit the 306 DS_1^{tricky} instances and the adjudication of reference errors and ambiguous specifications. The resulting evaluation partitions are reported in §3.

Audit procedure. The benchmark had previously undergone an audit in which two independent frontier models, blinded to one another, evaluated each reference against its natural language specification, followed by author adjudication of disagreements. A second, independent audit was then conducted over all 306 instances rather than only previously flagged rows.

Three frontier models from different model families independently generated a best-faith LTL formalization from each English specification and its declared atomic propositions. Each model also independently indicated whether the specification admitted multiple plausible interpretations. The models were not provided with the existing reference or the outputs of the other auditors.

Model agreement was used to identify cases for review, not as a correctness criterion. Eighty-four instances were flagged for adjudication. Manual review classified 15 as previously unidentified reference errors, 21 as cases in which the existing reference was already consistent with the specification, and 48 as specifications for which the available text did not determine a unique formalization. The 15 newly identified errors, together with 40 corrections from the earlier audit that were independently reconfirmed, give 55 confirmed reference corrections.

For each accepted correction, the adjudicated formula was checked against the natural language specification and compared with the original using BLACK's SAT-based LTL equivalence procedure. Solver checking was used to establish the formal relationship between formulas; the reference label itself was determined by semantic adjudication of the natural language requirement.

Representative corrections. The confirmed corrections span temporal scope, operator choice, formula structure, negation, counting, and proposition references. Representative cases are shown below.

- **n032 (scope).** The original reference `G((x1->(!x&X(x2)))&(x2->(!x1&X(x1))))` contains an undeclared atom and incorrectly scopes the first implication. The adjudicated formula is `G(x1->X(!x1&x2))&G(x2->(!x1&X(x1)))`.
- **n072 (temporal operator).** The original `G(x1 U F(x2))` introduces an additional eventuality operator inside the termination condition of the until expression. The adjudicated formula is `G(x1 U !x2)`, corresponding to the stated condition that x_1 holds until x_2 becomes false.
- **n014 (structure).** The original `G(x1->X(!x1 U x2))` does not encode the requirement that x_2 hold on the suffix following the last occurrence of x_1. The corrected formula is `G(x2 <-> G(!x1))`.
- **n056 (negation).** The reference contains `X(x3)` where the natural language requirement requires `X(!x3)`. The discrepancy is therefore a polarity error rather than a difference in temporal structure.
- **n133 (counting).** The reference does not correctly enforce the stated upper bound on the number of timestamps at which a is false. The corrected formula `!F(!a & X(F(!a & X(F(!a)))))` forbids a third such occurrence.

- **n033 (dropped condition).** The original formula captures only one condition from the natural language requirement. The corrected formula `F(x1) -> F(x2)` restores the missing "at least once" condition.
- **n199 (variable reference).** The original formula associates the until condition with the proposition representing upward motion rather than the proposition representing the passenger's request. The corrected formula associates the until condition with the latter proposition.

Ambiguity partition. A specification was retained as ambiguous when adjudication identified multiple non-equivalent formalizations supported by the wording and found no textual basis for preferring one interpretation.

For example, `n024` describes three people running a relay race: x_1 goes first, x_2 eventually goes next, and x_3 goes last until the race ends, with only one person running at a time. The audit identified distinct formalizations depending on whether the "eventually" relation is represented solely through the temporal ordering or accompanied by an explicit mutual-exclusion constraint. Both are consistent with the wording, and the specification does not provide a basis for selecting between them.

The 48 ambiguous specifications are retained as a separate partition rather than assigned a single adjudicated reference. Fifteen have been further categorized by ambiguity type (scope, quantification, or discourse structure); the remaining 33 were confirmed as non-resolvable during adjudication but were not assigned a finer subtype.

Unsatisfiable references. Six original references admit no satisfying trace. Five faithfully encode self-contradictory natural language requirements and are therefore retained. For example, a requirement stating that a proposition is always true and eventually becomes false is itself contradictory, and `G(x1&F(!x1))` faithfully represents that requirement.

The sixth unsatisfiable reference is among the 55 confirmed corrections. Its natural language requirement is satisfiable, whereas the original reference is not, and it was therefore corrected during adjudication.

Deliberate misconception items. Two benchmark instances are retained unchanged as deliberate misconception-test cases from prior work on common misunderstandings of LTL semantics (Greenman et al., 2022). These instances are intended to test the distinction between the stated requirement and a common but incorrect interpretation and are therefore not treated as reference defects.

E Selection Failure Analysis

This appendix characterizes the residual failures observed after candidate generation and semantic analysis. The analysis is qualitative: cases are grouped by the source of the observed error rather than re-estimating the aggregate results reported in §4. The categories are not mutually exclusive, since a single requirement can exhibit more than one failure mode.

Interpretation of temporal language. Several failures arise when the natural language requirement does not make the intended temporal scope explicit. For example, an "if X then Y" construction can correspond to a globally quantified requirement `G(X->Y)` or to a relation evaluated at a particular state `X->Y`, depending on the surrounding language. Similar distinctions arise for strong versus weak until and for eventual occurrence versus recurrence.

Representative requirements include "Whenever x_1 holds, x_2 holds as well," formalized as `G(x1 -> x2)`, and "If x_1 is true at the beginning then x_2 is true at the end," formalized as `G(x1 -> F x2)`. A third requirement, "If x_1's next state is true, x_2 will eventually be true," corresponds to `X(x1) -> F(x2)`. The common surface form does not by itself determine the same temporal scope in all three cases.

In such cases, symbolic analysis can distinguish the resulting formulas but does not provide an independent criterion for choosing between textually-supported interpretations. The observed failures therefore involve the interpretation of the requirement rather than a failure of LTL semantic checking.

Reference and evaluation artifacts. Some apparent selection failures in the original benchmark were attributable to the reference rather than to the selected formula. The audit procedure and the resulting corrections are documented in Appendix D. These cases are separated from the primary selection analysis rather than treated as model failures.

Vocabulary and grounding. A second class of failures occurs when the generated formula uses propositions that cannot be aligned reliably with the atomic propositions declared by the benchmark. The alignment analysis in Appendix G.1 documents these cases and the corresponding end-to-end measurements in Appendix B.8.

This failure is distinct from the temporal formalization itself: a formula can have the intended logical structure while remaining unevaluable under the benchmark's proposition vocabulary. The resulting error therefore arises at the interface between natural language propositions and the formal vocabulary rather than from LTL semantics.

Compiler-induced synthesis errors. A separate class arises at the interface between generated specifications and the specification-language compiler. Some generated constructions are syntactically valid to the language model but do not express the intended operator under the compiler's grammar. These failures are mechanically checkable and can therefore be addressed with explicit compiler guidance.

The controlled intervention reported in Table 11 provides the corresponding evidence: adding known specification-language pitfalls improves the affected synthesis path, whereas the interventions targeting semantic interpretation do not produce a comparable consistent effect. This provides a contrast between failures with an external mechanical criterion and failures whose resolution depends on interpreting the natural language requirement.

Candidate-set effects. Increasing the number of candidate formulas introduces another failure mode: a correct candidate can remain in the pool while a different candidate is selected. Pool-construction experiments examine this effect by introducing additional candidates through grounding, decomposition, FRET, and independent synthesis paths. The corresponding measurements are reported in Appendix B.7.

Inspection of these cases shows that additional candidates can change the relative ranking of otherwise unchanged candidates. The effect is therefore not equivalent to improving or degrading candidate generation alone; it changes the selection problem presented to the downstream judge.

Preprocessing failures. Requirement decomposition and lexical grounding introduce additional intermediate transformations before synthesis. In the evaluated configurations, both transformations produced cases in which an initially correct pipeline output was lost, as documented in the intervention results in Table 11.

These cases illustrate a separate failure mechanism from candidate selection: an intermediate representation can introduce an error before the candidate reaches the final selection stage. Consequently, adding another processing stage does not necessarily preserve the correctness of requirements already handled by the base pipeline.

F Implementation and Pipeline Details

This appendix specifies the implementation shared across the evaluated systems and the fixed control flow used by Pendulum. It describes the symbolic substrate, synthesis paths, selection procedure, and the unconstrained agent used in the comparative evaluation.

F.1 Symbolic Substrate

The evaluated systems access a common implementation of 14 tools through a single Model Context Protocol (MCP) server. The tool interface is therefore independent of the orchestration mechanism. Tool inputs and outputs use a common representation: formula arguments are strings, parser failures are returned as `invalid_params` errors with the underlying parser message, and formulas are serial-

Table 13: The 14 tools exposed through the common MCP server.

Tool	Input → Output
Symbolic tools	
`parse_and_canonicalize`	formula → canonical form, atomic propositions, NNF, simplified form, temporal class
`check_equivalence`	f_1, f_2 → equivalent?
`check_entailment`	f_1, f_2 → $f_1 \models f_2$?
`check_consistency`	formula(s) → satisfiable?
`compare_candidates`	candidates[] → pairwise equivalence/entailment relations
`gen_satisfying_trace`	formula → satisfying witness trace
`gen_violating_trace`	formula → violating trace
`distinguishing_trace`	f_1, f_2 → trace on which the formulas disagree
`check_trace_satisfaction`	formula, trace → satisfies?
`nl_to_ltl_via_salt`	SALT specification → LTL formula
`salt_help`	– → static SALT syntax reference
LLM-backed helper tools	
`extract_ap_mapping`	text → atomic-proposition mapping
`nl_to_ltl_via_python`	natural language → LTL formula
`ltl_to_nl`	formula → five natural language paraphrases

ized through a single canonical pretty-printer. Consequently, the output of one symbolic tool can be passed directly to another without an intermediate representation conversion.

Results are LRU-cached with capacity 1024. The three LLM-backed tools are cached only for temperature-zero executions with a fixed decoding seed. Symbolic tools are deterministic.

Shared semantics. All symbolic tools use the same semantics. Formulas are interpreted over infinite ω-traces. The next operator `X` uses the standard strong-next semantics, and the past operator `Y` is strict: `Y p` is false at the initial position. There is no separate weak yesterday operator. These semantics are implemented once in the shared Rust AST and are used by the downstream symbolic procedures rather than reimplemented separately by each tool.

Eight of the symbolic tools are backed by BLACK: equivalence, entailment, consistency, candidate comparison, satisfying-trace generation, violating-trace generation, distinguishing-trace generation, and trace satisfaction. These procedures reduce to calls to the BLACK propositional-LTL solver using the shared semantics above.

The trace-satisfaction tool has an additional implementation: a functional evaluator that traverses the typed AST directly over a finite witness trace. This implementation is used only as a cross-check of the solver-backed result. It is not used as the primary source of symbolic verdicts. The two implementations agreed on the development test suite.

SALT compiler. The `nl_to_ltl_via_salt` tool invokes a Dockerized build of the SALT specification-language compiler (Bauer et al., 2006). The input is a SALT specification and the output is either the LTL formula produced by the compiler or the compiler error. The compiler itself is deterministic and does not invoke a language model. Language-model inference enters the SALT path only when generating the SALT specification. Compiler errors can therefore be returned to the synthesis model as bounded mechanical feedback.

Optional pattern path. A pattern-lookup synthesis path based on the temporal specification patterns of Dwyer et al. (1999) is available as an independent synthesis seat. It is not enabled in the default Pendulum configuration. Its configuration and use in controlled experiments are documented in Appendix A.

F.2 FIXED PIPELINE

Pendulum is implemented as a LangGraph `StateGraph`. The control flow is fixed in code rather than selected by a language model. Each node is wrapped by a timeout-and-exception guard; a node failure is recorded in the run state rather than terminating the entire evaluation process.

The default topology is

$$\texttt{extract_aps} \rightarrow \{\texttt{synth_python} \parallel \texttt{synth_salt}\} \rightarrow \texttt{filter} \rightarrow \texttt{verify_all} \rightarrow \texttt{assess} \rightarrow \texttt{finalize}.$$

Pattern lookup, the BLACK-only deterministic verifier, and an LLM orchestrator finalizer are disabled in this configuration.

Atomic-proposition extraction. When the vocabulary is not supplied, `extract_aps` produces an atom-to-phrase mapping in one model completion. Malformed or low-confidence outputs are passed through one fallback extraction. If extraction still fails, the run terminates with an error status rather than continuing with an invalid vocabulary. When the vocabulary is supplied by the benchmark, extraction is skipped.

Candidate synthesis. The default pipeline contains two concurrent synthesis paths.

- **Code completion.** `synth_python` asks a coding model to complete an assignment to an LTL formula-AST constructor expression. The resulting expression is evaluated by a whitelist-restricted AST evaluator and then passed through `parse_and_canonicalize`. Disallowed constructs, including imports and arbitrary attribute access, are rejected. Parser failures are returned to the model for a bounded number of retries.
- **Specification compilation.** `synth_salt` retrieves relevant excerpts from the SALT manual, asks a language model to construct a SALT specification, and compiles the result using `nl_to_ltl_via_salt`. Compiler errors are returned to the model for a bounded repair loop. Every compiler output subsequently passes through the common parser gate.

The two paths synchronize before candidate filtering. Every candidate is parser-validated before entering the selection stage. Candidates are merged and deduplicated by canonical surface form. If the auxiliary filtering operation fails, the implementation falls back to retaining the candidate set rather than discarding the row.

Semantic analysis. For each surviving candidate, the verification stage constructs a common evidence record. The default configuration generates five natural language paraphrases using `ltl_to_nl`, one violating trace, and a pairwise candidate relation structure using `compare_candidates`. These operations are executed concurrently.

The resulting evidence, together with the natural language requirement, the atomic-proposition mapping, and each candidate's formula, is passed to a single assessment completion. The assessment produces a SUPPORTS or REFUTES verdict for each candidate. The controlled interventions in Appendix C vary this assessment stage by changing the model, adding additional evidence, or replacing the assessment procedure.

Feedback. Feedback is deterministic. If no surviving candidate receives a SUPPORTS verdict and the feedback budget has not been exhausted, the pipeline returns to synthesis with the assessment evidence. Otherwise, execution proceeds to finalization. The default feedback budget is one round.

Finalization. The default finalizer is deterministic. Candidates are ranked by the difference between their SUPPORTS and REFUTES counts, with ties resolved using the available confidence signal and then the synthesis re-emission confidence. Any candidate remaining tied after both signals is ordered by its position in the filtered candidate list, which reflects synthesis and deduplication order rather than an explicit rule. No additional language-model completion is required at this stage.

Every candidate was already validated by the parser at synthesis time (§F.2, Candidate synthesis), so the deterministic path returns the winning candidate's already-validated formula without re-parsing it.

Algorithm 1 Pendulum: fixed candidate-generation and selection workflow

Require: requirement s; atomic-proposition vocabulary V
Ensure: committed candidate $\hat{\varphi}$; surviving candidate set C

if V is not provided **then**
  $V \leftarrow \textsc{ExtractAPs}(s)$
end if
$C_{\text{code}} \leftarrow \textsc{SynthesizeCodeCompletion}(s, V)$
$C_{\text{spec}} \leftarrow \textsc{Compile}(\textsc{SynthesizeSALT}(s, V))$
$C \leftarrow \textsc{ParserGate}(C_{\text{code}} \cup C_{\text{spec}})$
$C \leftarrow \textsc{Deduplicate}(C)$
if $C = \emptyset$ **then**
  return FAIL
end if
for $\varphi_i \in C$ **do**
  $E_i \leftarrow \text{BLACK}(\varphi_i, C)$
end for
$\{v_i\}_{\varphi_i \in C} \leftarrow \textsc{Judge}(s, C, \{E_i\})$
$\hat{\varphi} \leftarrow \arg\max_{\varphi_i \in C} (\mathbb{1}[v_i = \textsc{Supports}] - \mathbb{1}[v_i = \textsc{Refutes}])$
return $\hat{\varphi}, C$

Re-parsing at the finalize boundary is used only by the non-default LLM-ranking finalizer, which discards any returned formula that fails to parse.

Pendulum returns the committed candidate and the surviving candidate set. Evaluation against the audited reference is performed outside the pipeline; the system therefore does not require access to the evaluation reference during execution.

F.3 Unconstrained Agent

The unconstrained agent provides a separate orchestration baseline. It is a single language-model controller operating over a bounded ReAct-style loop. The agent reuses the two synthesis modules and the MCP client from Pendulum without modification, while using an independent model seat. Thus, the comparison changes orchestration rather than the underlying synthesis implementations.

Tool protocol. The evaluated backends expose heterogeneous interfaces: local Ollama executions return free text, while the command-line clients used for the frontier models are prompt-based. The agent therefore receives tool signatures in its system prompt rather than a provider-specific tool-calling schema. Each action is represented as a JSON object containing either a tool name and arguments or a final candidate list. The protocol parser accepts the equivalent fenced and unfenced forms so that malformed protocol syntax is not conflated with a formalization error.

Tool surface. The default agent exposes the two synthesis sub-agents, `parse_and_canonicalize`, `ltl_to_nl`, `distinguishing_trace`, `gen_violating_trace`, `check_trace_satisfaction`, `compare_candidates`, and `check_equivalence`. Atomic-proposition extraction can additionally be exposed when the vocabulary is not supplied. The tool set can be restricted by configuration for ablation experiments.

The synthesis sub-agents are the same implementations used by Pendulum; they are not replaced by the raw MCP translation tools. This keeps the synthesis mechanism fixed when orchestration is compared.

Candidate state. All candidates produced during the agent's execution are retained in run state and deduplicated by canonical form. The agent may reorder these candidates at finalization but cannot discard a candidate solely because it was generated earlier in the run. This makes the candidate pool available for coverage measurement independently of the agent's final decision.

Verification gate. Before accepting a final answer, the agent is required to invoke at least one sentence-facing analysis tool on the selected candidate. The gate accepts `ltl_to_nl` and the trace-

analysis tools. Pairwise equivalence and candidate-comparison tools alone do not satisfy the gate because they characterize relationships between formal candidates rather than their correspondence to the natural language requirement.

Budget and termination. The agent is limited to 15 tool calls and a wall-clock budget, with a hard turn limit above both. Every termination path invokes finalization, including a safe finalization path used when the MCP server fails. Repeated empty actions, identical-call repetition, unchecked final answers, and unparseable outputs are each subject to bounded corrective prompts. When a corrective budget is exhausted, the agent finalizes using the candidates already present in its run state.

Additional prompt-level exploration experiments are reported with the orchestration results in Appendix B.9; they are not part of the implementation specification above.

G Scoring and Statistical Analysis

G.1 Alignment Scorer

When a system extracts its own atomic-proposition vocabulary, its atom names may differ from the benchmark placeholders. For example, a system may produce `x1_true` or `rain` where the benchmark uses `x1`. Verbatim comparison would therefore mark a structurally correct formula as incorrect even when the system has identified the corresponding proposition. Rows with an incorrect number of propositions remain failures; the alignment procedure addresses only the naming mismatch.

Why the built-in AP correspondence is insufficient. The `check_equivalence` tool can compare formulas modulo a supplied atomic-proposition correspondence, but the correspondence requires the system-side description to match the benchmark gloss exactly. This provides little useful grounding when benchmark propositions are represented by opaque placeholders. An independently generated vocabulary therefore requires a separate correspondence procedure before semantic equivalence can be evaluated.

Construction. The alignment scorer evaluates candidate correspondences between the system-generated and benchmark vocabularies using a fixed priority order. The evidence sources are exact fragment matches, benchmark atom names appearing in system-generated atom names, overlap between benchmark glosses and system descriptions, and token overlap with light stemming. The highest scoring assignment is accepted only when every matched pair has non-zero evidence and the resulting assignment is unique. Otherwise, the row is marked as ungroundable and retains its strict, verbatim verdict.

The uniqueness requirement is important. Without it, weak textual overlap could produce an arbitrary correspondence between semantically unrelated propositions and artificially increase the measured equivalence rate. Once an assignment is accepted, the system formula is renamed according to the bijection and evaluated using the same solver-backed equivalence procedure used for supplied-vocabulary results.

Validation. The scorer is validated on the supplied-vocabulary condition, in which the system already uses the benchmark's atomic-proposition names. The alignment procedure leaves the results of this condition unchanged. This provides a control against score increases caused by the alignment procedure itself.

Ungroundable rows. Some system-generated vocabularies cannot be aligned to the benchmark placeholders using textual evidence. This occurs, for example, when a system assigns a semantically meaningful name such as `catherine_lounge` to a proposition represented only as `x3` in the benchmark. Such rows retain their strict verdict rather than receiving a speculative correspondence. Consequently, aligned results reported in Appendix B.8 are conservative lower bounds on performance in the absence of a supplied vocabulary.

G.2 Metric Definitions and Statistical Tests

Candidate coverage $any - of(s)$ and candidate selection $top - 1(s)$ are defined in §2.1. Both are evaluated against the audited reference φ^* using solver-backed logical equivalence; textual similarity is not used to determine correctness. Pendulum itself returns only the surviving candidate set and the committed candidate. Evaluation against φ^* is performed afterward by the evaluation harness.

For paired binary comparisons, the paper uses the exact McNemar test. We report the two discordant counts: b for cases in which the first condition is incorrect and the second is correct, and c for cases in which the first condition is correct and the second is incorrect. Proportions are reported with Wilson confidence intervals where applicable. Rank-graded comparisons (linear, MRR, NDCG), where reported (Appendix B), use a paired bootstrap with 10,000 resamples and a fixed seed.

For controlled interventions, the primary comparison is a paired regression check. The evaluation counts requirements corrected by the intervention and requirements that regress under the same comparison set. The reported net effect is the number improved minus the number regressed. This prevents an intervention from being described as beneficial solely because it fixes a small number of additional requirements while silently breaking previously correct cases.

The complete paired results, discordant counts, and confidence intervals for the reported comparisons are provided in Appendix B.