Check Before You Commit: Selective, Puzzle-Internal Verification on LingOly-TOO

Peiyang Lin
Kang Chiao International School, Linkou Campus
Taiwan
peiyanglin2027@gmail.com

Chunki Lim
National Taiwan University
Taiwan
r14142001@ntu.edu.tw

Abstract

Large language models often produce plausible but incorrect answers to linguistics-olympiad puzzles, and obfuscated benchmarks such as LingOly-TOO make such errors hard to detect. We introduce a modular harness that converts a model's proposed hypothesis into a deterministic, puzzle-internal test: the model proposes a mechanical skeleton for a puzzle, and deterministic code solves and verifies that skeleton against the puzzle's own given examples, committing an answer only when verified and abstaining otherwise. On the development problems, the system commits with 97.7% precision, with 128 correct commits of 131 at 10.5% coverage. On 50 unseen problems, the system makes 73 commits with 35.6% precision and 47 errors, 46 of which were clustered around one problem whose givens do not determine a single rule. Three case studies characterize the system: deterministic solving of a scrambled-numeral puzzle with no model calls, safe rejection of an incorrect model proposal, and a verifier failure in which a rule reproduces every given yet generalizes wrongly to the query. Re-fitting the rule under leave-one-out separated correct from incorrect commits on the two problems tested. We find that verification is exact for deterministic solvers, while for model-aided solvers it is necessary but insufficient.

Keywords: linguistics-olympiad, selective prediction, verification, large language models, LingOly-TOO

1 Introduction

Current frontier models achieve non-trivial but far-from-solved scores on linguistics-olympiad puzzles (Bean et al., 2024; Goyal and Dan, 2025). However, prior research already demonstrates through obfuscating the puzzles that a substantial proportion of the score can be attributed to memorization, as the best reasoning models drop from 0.59 to 0.48 exact-match after obfuscation (Khouja et al., 2026). These studies mainly focused on implementing robustness to memorization at the benchmark level, not the per-answer level. Namely, they provide a single aggregate score, but are unable to determine whether the response was achieved via reasoning or guessing. On the individual level within a puzzle, a confident-but-wrong solution is strictly worse than an abstention, raising the issue of determining when the model knows it is correct as opposed to a flat aggregate score. Model-driven auto-rectification without an external signal is unreliable, as models have been shown to not be able to demonstrably fix their own reasoning chains (Huang et al., 2024), but achieve accuracy gains when primarily paired with reliable external feedback (Kamoi et al., 2024). Generic self-refinement employs the same model for generation, refinement, and judgment without an objective standard (Madaan et al., 2023). Furthermore, aggregated verifiers are approximate and cannot certify (Saad-Falcon et al., 2025), allowing a wrong answer to still pass through sporadically. Preceding procedures lack an external, exact correctness signal, which a self-contained linguistics-olympiad puzzle uniquely provides. We restrict the model to solely proposing a mechanical skeleton (string segmentation or which dimension a pattern varies on) in lieu of a rule or direct answer. We then employ deterministic code which carries out the inference and checks it against the puzzle's own givens via leave-one-out, either failing or passing this gate. Only after passing the final check does the system commit the analysis, alternatively abstaining to a plain baseline answer. Our approach contrasts prior frameworks by checking whether the proposed solution is exact on the givens; the rule either reproduces the puzzle's initial data or

Proceedings of the 38th Conference on Computational Linguistics and Speech Processing (ROCLING 2026), pages 371–382
November 6-7, 2026 ©2026 Association for Computational Linguistics

fails to do so, eliminating the approximation error in the middle. Moreover, this selective-prediction mechanism (Wen et al., 2025) is applied puzzle-internally on a cell-by-cell basis: general-coverage is traded for elevated precision, and the confidence signal is re-derived each instance from the givens themselves, rather than a global learned score (Kamath et al., 2020). This framework follows similar systems in which the model proposes the structure while deterministic code produces the response (Wang et al., 2024; Gao et al., 2023; Lyu et al., 2023), but preceding methodologies commit to whatever the code returns, and are bound to always emit an answer. We take the approach of adding a leave-one-out gate, hiding a given and checking that the other givens still support its derivation, abstaining if they do not. As such, inference is moved onto the deterministic layer, and the model is never blindly trusted to perform verification on its own—the analysis is verified directly against the puzzle. We evaluate this on LingOly-TOO with Gemini 2.5 Flash.

Contributions.

- A commit-or-abstain harness built on a four-part leave-one-out gate: reproduce all givens, satisfy leave-one-out, be determinate, and cover the queried cell.
- Five solver-verifier modules: numerals, morphology, translation, vowel harmony, and a permutation CSP. 110 of 131 committed answers were produced with zero model calls, ensuring they cannot be attributed to the memorization of the puzzle.
- A map of the verification frontier across the International Linguistics Olympiad (IOL) problem families our gate solves-and-verifies, and the ones it abstains on (Figure 2).
- A leave-one-out ablation swaps the LOO check for a full-fit check, verifying against all givens on the dev split; applying a full-fit gate changes net commits from 131 to 132, while exact cells stay 128 (precision drops from 0.977 to 0.970). Full-fit yields no net gain of correct answers but introduces an error.
- An evaluation of the frozen system on a set of 50 unseen problems, with rules pre-registered prior to the run, and a re-fit leave-one-out examination, which together mark the flaws of the method—model-assisted rules on puzzles whose givens do not isolate a rule.

2 Background and Related Work

Inducing structure from a puzzle's givens. A lineage of work aimed to study various broad reasoning tasks, focusing on how models propose a symbolic form or program, which deterministic code executes. A paradigm that has been tested on various inductive-reasoning benchmarks such as ARC utilizes an LLM to propose candidate hypotheses, summarize them, implement them as programs, then run them to verify whether they are able to replicate the given input-output examples (Wang et al., 2024). Earlier work examined offloading an LLM's reasoning chain to a Python program, with the interpreter doing the computation when solving math word problems (Gao et al., 2023). Other methods include converting a query by interweaving natural language with symbolic steps (Python/Datalog/PDDL) executed by a deterministic solver (Lyu et al., 2023), or translating the problem into a symbolic form with an LLM, which an external symbolic solver runs and refines from (Pan et al., 2023). Separately, research studies LLMs and their abilities to solve linguistics-olympiad puzzles specifically: benchmarks measuring LLM performance on Rosetta-style puzzles (requiring the inducing of a rule from a few given pairs) (Şahin et al., 2020; Majmudar et al., 2026), analyses of the origins of puzzle difficulty (Choudhary et al., 2025), and work on both generating and solving these puzzles (Majmudar and Filatova, 2025). One such analysis isolates morphological segmentation as a bottleneck for LLM performance on these puzzles (Choudhary et al., 2025). Our method takes the broader process of the former propose-compute split and applies it within the field of linguistics-olympiad puzzles. Regarding our modules, we build on existing approaches for solving morphology (Kann et al., 2020; Creutz and Lagus, 2007), and numeral (Maier and Wolff, 2023; Rubehn et al., 2025) puzzles. However, whereas contemporary systems typically tune the rule until it fits across all givens, at which point it becomes trusted by the system, we add a leave-one-out gate: we first fit the rule once across all givens, then check whether each given it reproduces is also supported by the others, abstaining if it does not. Hypothesis Search, the most directly comparable approach to our own, checks its

programs on observed examples, but does not implement any abstention mechanism (Wang et al., 2024). Thus, we append an abstention state rather than always committing by default.

Self-checking and learned verifiers. A large body of work relies on the model self-revising its own answers, but LLMs do not reliably rectify reasoning shortfalls when not exposed to external feedback (Huang et al., 2024), and self-correction has been found to be most beneficial when paired with reliable external feedback (Kamoi et al., 2024). These methods have traditionally clustered around refinement using external feedback or tools, or using the same model as its own critic (Shinn et al., 2023; Gou et al., 2024; Madaan et al., 2023). A distinct study establishes an aggregate verifier through combining many weak and imperfect ones, narrowing the gap to an oracle verifier (ideal verifier) but not reaching it (Saad-Falcon et al., 2025). Across all these configurations, the check is either the model self-judging without any external reference, or a learned, approximate verifier. A self-contained puzzle uniquely supplies an exact objective signal with which to compare hypotheses through its givens; our system makes use of this fact and implements the LOO gate, providing it with a clear, exact outside signal.

Obfuscated benchmarks. Contamination has been found to inflate the perceived performance of the reasoning of the model, leading to work quantifying contamination separately per benchmark (Sainz et al., 2023). Detection remains fragile, however, as it can be bypassed by rephrasing items (Yang et al., 2023), motivating the creation of continually-refreshed benchmarks (White et al., 2025). Within the field of linguistics-olympiad evaluation itself, LingOly-TOO obfuscates orthography to block model memorization of puzzle languages (Khouja et al., 2026), producing a measurable effect, with leading reasoning models falling from roughly 0.59 down to 0.48 post-obfuscation, driving us to evaluate on the latter obfuscated set. Lian et al. (2025) creates a structured linguistics-olympiad benchmark (LOBSTER), comprising granular evaluations beyond final-answer accuracy spanning over 90 languages. It identifies orthographic obfuscation as an approach for alleviating the contamination it flags, pointing to it as future work. The obfuscated set this paper evaluates on continues in that direction. As opposed to existing contamination controls, which focus on curating, rotating, and obfuscating items within the dataset to control for memorization, we attempt to determine whether a single obfuscated puzzle's correctness can be established from its givens alone, abstaining when it fails to do so.

Knowing when to abstain. Abstention is a well-documented approach, including systems abstaining below a certain confidence threshold (Geifman and El-Yaniv, 2017), detecting a distribution disparate from training (Kamath et al., 2020), or simply reporting when it does not know (Zhang et al., 2024). The space itself has been broadly surveyed (Wen et al., 2025). Our abstention mechanism differs in basis for decision-making, as existing ones rely on a confidence or uncertainty estimate, while we employ an exact puzzle-internal check (leave-one-out reproduction on the givens) to decide whether to abstain.

3 Method

3.1 Model proposes, code verifies

Proposal. Within our pipeline, the LLM calls function solely for proposing the mechanical skeletons (how each string segments or tokenizes, or on which property a pattern varies upon), and never the rule or answer values. Deterministic code forms the inference and verification layers.

Verification. Structuring the system in this manner allows it to either commit or abstain through a four-part deterministic gate (see Section 3.2 and Figure 1): the induced solution must reproduce all of the puzzle's given rows exactly, satisfy leave-one-out (hereinafter abbreviated LOO) on the givens, be determinate (no two admissible analyses disagree on the committed span, the portion of the output the system fills for the queried cell), and cover the queried cell, namely, it produces an answer for the cell. The pipeline commits to its solution only if all four pass; otherwise it abstains, falling back to the freeform baseline. Choosing to commit only when the system can reliably verify that the rule is accurate makes committed output match or exceed the baseline in expectation (commit precision far exceeds baseline accuracy), so committing is additive on average, though a false commit can lose on its own cell (see Section 5.3). The remaining question is which hypothesis classes the code itself can verify, five of which are enumerated in the following section.

Avoiding per-language rules. The verifier mod-

ules apply general procedures, notably segmentation search, condition induction, constraint solving, and finite-answer matching, with no puzzle- or language-specific rules or values hard coded in the solving path. Thus, the same code functions identically across every problem.

3.2 The verification gate

Gate. The gate's main purpose is to maintain the inductive premise: a rule that reproduces the givens while they are hidden is generalizing instead of memorizing. A naive rule, fit to all rows at once, cannot distinguish between the two. This is where our verification gate differs from structure-induction ancestors (see Section 2). We enforce it with four sub-rules: (i) reproduce: the output must satisfy reproduce == 1.0 on every given shown; (ii) LOO: each held-out given must be derivable from the rule, with every part of its derivation (stem, affix branch, and rewrite) also found in another given; the rule is fit once from all givens and not subsequently re-fit per fold. M2 commits at a LOO score of $\geq$0.9, and every analysis committed in our runs scored 1.0. When withdrawing a row leaves an analysis with no support, that fold is untestable and is exempted under atomic-witness; if no fold at all is testable, the system abstains rather than accepting a vacuous pass; (iii) determinacy: the gate requires at least two supporting examples, and if two admissible analyses disagree on the committed span, it abstains; (iv) cover: the rule produces an answer for the queried cell. The solver is gold-blind throughout: the solver never reads the gold answers, only the givens, and every reproduction check is compared after NFC normalization on both sides. Maintaining LOO, not a naive full-coverage fit, is intended to separate rules that generalize to the unseen query from rules memorized based on the shown rows; Section 5.1 shows where this check proves insufficient. Section 3.3 applies these criteria per subtype across five modules.

3.3 Verifier modules

The gate acts as the verifier for the five modules, each targeting one hypothesis class, which we label M1 through M5 (Table 1). Within its class, the module takes the LLM's proposed mechanical skeleton, constructs the candidate analysis in code, and runs the class-specific LOO check before it is allowed to commit. M1 (numerals): the model proposes how a numeral string segments, and the code enumerates candidate frames and verifies each against the givens; M2 (morphology): the model proposes the stems, and the code induces the conditioning rule via a decision-list search and verifies under LOO; M3 (translation): the model proposes nothing rule-level, and the code aligns the given pairs and resolves the query as a matching problem over a finite candidate set fixed by the givens; M4 (vowel harmony): the puzzle states that N listed words are loanword exceptions, and the model proposes the harmony dimension while the code commits only when exactly N words violate affix-root harmony and every other word is harmonic, the committed answer being those N loanwords; M5 (permutation): the scrambled-numeral constraint-satisfaction problem (CSP) is solved and verified end-to-end in code. Two of the five modules (M3, M5) make zero model calls, and the code both forms and verifies the inference, and thus the core inference resides in the deterministic layer instead of the model. Section 5.1 (Panel A) quantifies what share of committed answers comes from these zero-call modules. Section 3.4 covers how the committing module is selected and how the system falls back to the baseline when abstaining. Though, the five modules remain proofs of concept and not an exhaustive solver: phenomena unaccounted for by any covered class (reduplication, multi-slot clitics) are abstained on.

3.4 Commit-or-abstain composition

We run each module concurrently on every cell and answer-blind (it only sees the puzzle's public view and never the gold). Each module self-gates: it either commits (all four Section 3.2 conditions pass) or abstains. Thus, nothing has to decide which module to run, and they all try. The final committed solution is resolved per cell by commit count: 0 commits lead to a fallback to the baseline response; 1 commit leads to the solution being taken; $\geq$ 2 commits take the highest LOO score (in the scenario of an exact LOO score tie, the gate enacts a tie-break and falls back to a fixed order, M1 > M2 > M3 > M4 > M5, ensuring reproducible results independent of thread scheduling). The system logs every collision, and because modules target disjoint classes, collisions are rare by design. The per-module gate drives precision, as each analysis needs to be committed by the gate, and not a classifier; misrouting is impossible because a module cannot commit to a cell it cannot verify. The committed and abstained cells are then

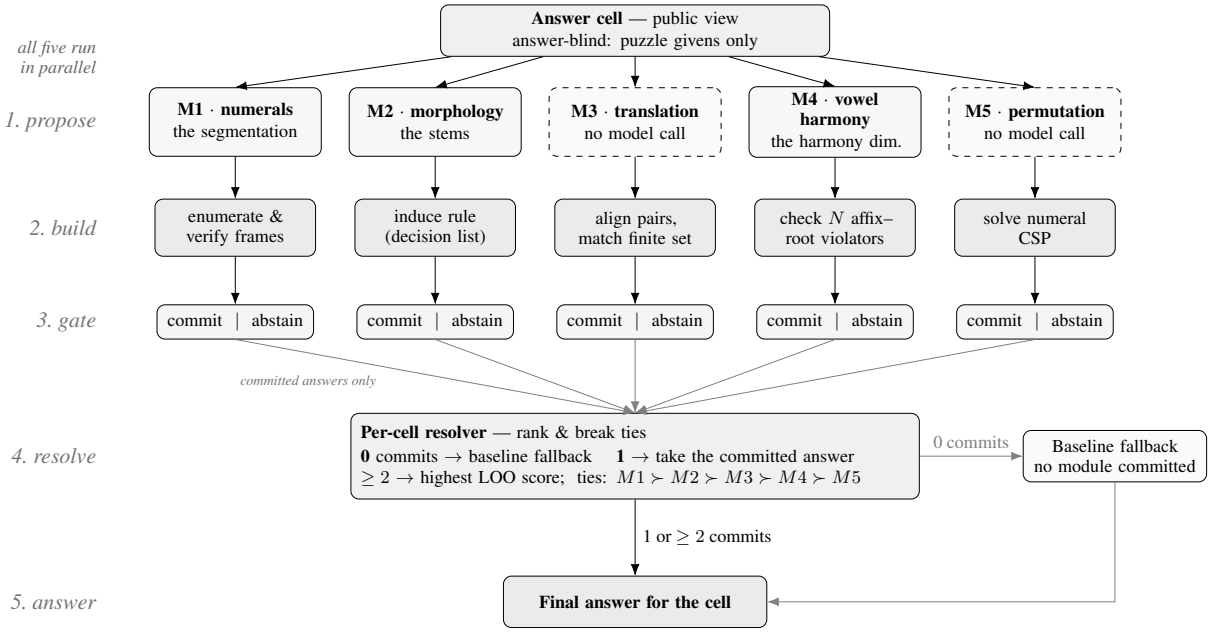

Figure 1: System architecture. All five modules attempt every answer cell concurrently and answer-blind; there is no router and no ordering in execution. Each module proposes only a mechanical skeleton, builds a candidate with deterministic code, and applies its own four-part gate (reproduce the givens, leave-one-out, determinacy, and cover the queried cell; Section 3.2), independently committing or abstaining. Two modules (M3, M5, dashed) make no model call. A per-cell resolver then combines the five decisions by commit count: no commit falls back to the baseline, a single commit is taken, and ties among two or more are broken by highest leave-one-out score, then by the fixed order $M1 \succ M2 \succ M3 \succ M4 \succ M5$. Applicability is decided by verification, not by a classifier, so a module cannot commit to a cell it cannot verify.

assembled in a single submission.

4 Experimental Setup

Data and splits. LingOly-TOO (Khouja et al., 2026) refers to a benchmark of orthographically obfuscated linguistics-olympiad problems, with each problem appearing as ∼6 obfuscated variants (p=1–6) alongside the original (82 problems, 1203 questions, 6995 sub-questions). We evaluate a subset of it, 32 of 82 problems (39%), 200 of 1203 questions (16.6%), 1245 of 6995 sub-questions (17.8%), with 160 variants, capped at 5 per problem. Our method is dev-guided: we inspect why a problem fails and build a module for that shape, thus any problem iterated on is permanently spent for evaluation. The remaining problems were never loaded throughout our evaluation. The split was along individual problems, not the rows, ensuring induced grammar does not contaminate inter-set. The 200 rows are exactly the first 200 rows of the released LingOly-TOO test file, so the subset follows file order. It could not have been selected based on performance, as it was selected and fixed on 2026-07-10, before the development of M1 (2026-07-12) and before the first reported run (2026-07-15). 24 of the 200 rows (169 of 1245 cells, 13.6%) are unobfuscated originals (p=0), found in 20 of 32 problems. Across the set we evaluated, we differentiated between two splits: the dev split consisting of 825 cells across 23 problems, and the held-out split consisting of 420 cells across 9 problems. The final distribution was frozen 2026-07-15. Originally, 11 problems were planned to be reserved and never run or inspected, but problems 131 and 169 were folded into dev following the m3_match v2 build. Neither merged problem contributed any committed cells (Section 5.1), leading to zero change to the reported number. The remaining 9 received an initial frozen diagnostic run per relevant module, and were never subsequently fitted on. Problem families were hand-classified by the authors, as LingOly-TOO ships no task-type metadata, based on the underlying phenomenon of each question.

Gold isolation. Gold values are confined to scorer files, distinct from the solver core, which remains gold-blind throughout: inputs for the solver modules include only the setup of each row (prompt, context, and question fields), so gold val-

ues are quarantined from the solving path, and we use a bit-identity test to confirm solver behavior remains constant when gold values are withheld. We also perform Unicode NFC normalization on both sides of every reproduction check to remove any incidental mismatches caused by formatting.

Model and caching. We run the harness on Gemini 2.5 Flash, which modules M1, M2, and M4 query; M3 and M5 invoke no model calls. The results are reported through a warm-cache replay, with the calls issued once during development and the cached completions served at report time. There were no model calls during the report (no live model calls) and no cache misses in the reported run. Across each module, the total number of requested model calls was 695 (M1 **15**, M2 **675**, M4 **5**; M3&M5 0). A further 6 calls were cap-denied (no completion returned), affecting 4 rows (12 cells) on problems 20 and 24. Those cells are counted as abstentions by default; we cannot know what the gate would have decided had the calls returned.

Fresh held-out set. For the new run, we used the remaining 50 problems not present in the dev set and all their questions, for a total of 692 questions and 4,307 cells. We ensured the code for the system was frozen by saving and tagging it prior to the run. The rules for the run were also written beforehand, including mandating the reporting of any results. The model calls for the run totaled 2,257, all coming from M2. We note one interruption during the run, due to a premature depletion of API credits resulting in a pause of the run, which was then resumed from saved responses. One call failed, so we present the question as an abstention. The reported numbers were obtained by replaying the saved responses without additional model calls.

Metrics. There are two main scoreboards: Panel A (primary), which records the selective commit precision at coverage, specifically, of the cells the harness commits to, what fraction are exact, and what is the overall coverage; Panel B (secondary), the composed-vs-baseline exact-match lift, a whole-question exact-match (EM) of the harness submission minus the baseline's, which is limited in coverage by design. Panel A serves as the primary contribution, while Panel B is reported as a secondary check.

5 Results

5.1 Selective precision (Panel A)

Headline. Across 1245 cells, the harness commits 131, of which 128 are exact, a precision of 0.977 at 10.5% coverage (89.5% abstain). On dev alone, the part the harness was fitted upon, the same 131 commits cover 15.9% of the 825 cells. On these same 131 cells, the freeform baseline is exact on only 32 (24.4%), so committing replaces mostly-wrong baseline answers. The coverage is reported over all evaluated cells throughout, matching the figure of the abstract.

Fresh held-out evaluation. On the original run, the system committed nothing across the original 9 held-out problems (0 commits across 420 cells; forcing commits on the 125 under-determined cells gives 8.0% exact, Appendix B). On a new 50-problem set (4,307 cells), unseen by the system prior to evaluation, it made 73 commits, 26 of which were exact. This shows a lowered precision rate at 35.6% (95% CI 25.6–47.1) at 1.7% coverage (Table 1). All of the commits came from M2, with the other modules abstaining completely, M1 in particular rejecting the numeral problems as an unsupported problem shape. Of the 47 wrong commits, 46 came from Problem 91, though each commit had a perfect LOO score. As an example, the system committed `foohalol` whereas the gold answer was `foohadol`. This is the same failure of the system as encountered in Problem 139, though at a larger scale. Thus, the 97.7% precision holds only for the dev-split, and the weakness is specific, and can be traced back to model-assisted rule induction on puzzles whose givens don't narrow down the rule.

Division of labor. 110 of 131 commits (M3's 60 + M5's 50) come from zero-model-call modules and are all exact. The remaining 21 commits come from model-calling modules (M1, M2, M4), 18 of which are exact (85.7% precision, 95% CI 65.4–95.0). The deterministic modules drive the 97.7% precision. A module-level replication under a second model is reported in Appendix D. Leakage of a held-out given into the model's proposal can only affect the model-calling modules, as the majority (110 of 131) of commits were produced by M3 and M5, which make zero model calls and thus cannot be affected. Of the 21 model-assisted commits, we re-ran the 18 M2 commits with each given removed before proposing and inducing (re-fit leave-one-out, below); the 3 M1 and M4 commits were

not re-tested.

Coverage as a property of the gate. The low coverage is by-design, explicable when accounting for a strict gate and a limited hypothesis-class library. These results do not demonstrate under-testing as all 160 variants were driven. As expected, the gate declines whole families it cannot verify rather than guessing; relaxing it would recover nothing, as hypotheses that reproduce 50–99% of the givens answer zero query cells correctly (Appendix C).

Leave-one-out versus full-fit. Replacing LOO with a full-fit check (verify analysis against all givens, removing held-out cross-check) on the dev-split (825 cells, 23 problems) commits additional three cells on variant 155_0000, 2 correct and 1 wrong, while losing 2 correct commits on variant 155_0005. This increases the net commits to 132 cells, while the exact count remains 128 cells (precision falls from 0.977 to 0.970). Full-fit thus yields no net gain of correct answers and introduces one error. The entire net effect is one cell, however, thus bounding the gate's contribution rather than demonstrating it drives the result. M2 output identically under both settings, so the whole effect is M1 on problem 155.

Re-fit leave-one-out. For the 18 M2 commits, we individually removed each given, then asked the model for the stems, verifying if it can rebuild the rule and reproduce the removed given. In Problem 135 (15 commits, all correct), the rebuilt rule reproduced 34–35 of the 44 testable givens for each variant. The committed answers were also recurring across 41–43 of the 44 runs, while the misses were irregular forms that were unpredictable from the givens, such as `mangar nar`. In Problem 139 (3 wrong commits), however, the rebuilt rule wasn't able to reproduce any of the testable givens. While the system passed our original LOO check in both problems, scoring a perfect 1.0, rebuilding the rule helps clearly differentiate between correct and incorrect solutions. As our sample size is small, only two problems of the set, we hesitate to propose a cutoff score. Instead, we recommend future work focusing on rebuilding the rule as a way of strengthening the gate.

5.2 The commit frontier

Commit distribution. Commits are classified by phenomenon and concentrate in Numerals & Calendar (52 exact, 0 wrong) and Morphology (75 exact, 3 wrong) problems. Phonology & Writing had 1 commit, while Syntax and Semantics & Kinship both had 0. The distinct families label the phenomenon present on each puzzle and are orthogonal to the module axis of Table 1; a module can commit independently of its intended family (all 60 M3 commits fall in problems classified as Morphology). These results reflect the gate's strictness, as it declines entire families (Syntax, Semantics) it cannot verify rather than committing inconclusive analyses.

Aggregate lift (Panel B). Layering committed cells over the baseline raises whole-question exact match by 3.65 points (0.5352 vs 0.4987; 95% paired-bootstrap CI [1.6, 6.1] points). Both submissions are scored under the same normalized key (whitespace-stripped, NFC), applied identically to the composed and baseline submissions; the baseline is 0.4937 under the strict exact key.

Module	Commit	Exact	Prec.	95% CI
M1 (numerals)	2	2	1.00	0.34–1.00
M2 (morphology)	18	15	0.83	0.61–0.94
M3 (translation)	60	60	1.00	0.94–1.00
M4 (harmony)	1	1	1.00	0.21–1.00
M5 (perm. CSP)	50	50	1.00	0.93–1.00
All	**131**	**128**	**0.977**	0.93–0.99
M3+M5 only	110	110	1.00	0.97–1.00
Fresh (50 problems)	73	26	0.356	0.26–0.47

Table 1: Panel A: selective commit precision by module across the 32-problem set (Wilson 95% intervals). Held-out (420 cells): 0 committed. The fresh row is the frozen system on the 50 problems never used in development (4,307 cells; all 73 commits from M2). Per-module intervals rest on few commits and should be read with caution.

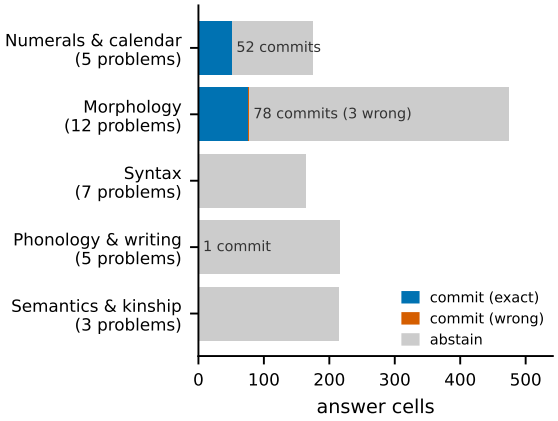

Figure 2: Frontier map of commit concentration in puzzle families across 32 problems. Syntax and Semantics & Kinship saw zero commits.

This finding is attributable to a lift bounded by coverage by construction, as abstained cells scored as the baseline, and the commits concentrated almost entirely in one problem (scrambled-numeral permutation). These results thus do not constitute evidence of broad end-to-end improvement, and instead maintain the selective precision of Section 5.1 as its core contribution. M5 runs answer-blind on all 1245 cells, and was not routed to problem 168, but commits the 50 problem 168 cells (all exact) and makes zero false commits on every other cell, where it is incompatible. As a result, rather than being due to a solver built for one problem, the concentration shows the feasibility of employing a gate to limit each module to where LOO verifies without dependency on a router.

5.3 Case studies

Deterministic solving (Problem 168). Problem 168 (Table 2) is a scrambled-numeral puzzle, where the givens are numeral expressions whose symbol-to-value mappings have been obfuscated. M5 does not propose any analyses, and instead makes use of the problem structure being fully determined by the givens, making it a constraint-satisfaction problem, in which each equation constrains symbol-to-value mapping. M5 enumerates the permutations consistent with all givens, allowing it to solve the puzzle with zero model-calls; it has no linguistic hypothesis to guess, only a mapping forced by the givens. For each instance of the problem, only a single permutation satisfies every given; consequently, the recovered mapping is unique, and when applied to the five instances of problem 168 (50 cells) M5 is exact on all 50. When a puzzle's structure is fully forced by its givens, verification and solving collapse into a single deterministic step, circumventing model calls and any risks associated with them.

Safe rejection. Problem 5 (Table 2) is a vowel-harmony puzzle with 5 instances. M4 proposed the harmony analysis, and it was applicable on the first sub-questions across all 5 of the variants, with the gate verifying whether they were able to reproduce the givens; it committed on one (variant 5_0004, sub-question a) and abstained on the remaining 4, as the logs reveal the proposals reproduced zero givens (m4_abstain: count=0). The one commit is the single entry in Panel A, and the case shows the precision the system gains from verification instead of implicit confidence in the model's analyses.

Problem	Mod.	Calls	Cells	Commit	Exact
168 (numerals)	M5	0	50	50	50
5 (harmony)	M4	5	5	1	1
139 (morphology)	M2	15	50	3	0
91 (fresh)	M2	19	105	69	23

Table 2: Case studies. Calls: model calls by the committing module; Cells: cells the module attempted (for Problem 5, sub-question a of each variant). 168 shows deterministic solving, 5 safe rejection, and 139 a verifier failure: the rule reproduces every given at leave-one-out = 1.00 yet commits `uingani` for gold `engani` (Appendix A). Problem 91, from the fresh set, shows the same failure on unseen data.

Verifier failure. Problem 139 is a morphology puzzle, M2's class, with 5 variants across 50 cells (Table 2). M2 serves to induce a rule that reproduces every given and survives LOO (LOO = 1.00), which it did, passing all four gate conditions. The commit was a partial span (m2_committed_partial), given that it only filled part of the queried cell. It committed on sub-question (d) in three of the five variants (139_0001, 0002, 0003), abstaining on (d) for the other two. All three commits were erroneous, forming the only false commits in the development run. A perfect verifier score is necessary but not sufficient for all cases; the failure is concentrated on one query type, sub-question (d), that the givens never constrained, thus the same induced rule generalizes wrongly while simultaneously the results of all rows validate successfully. This presents the ceiling of puzzle-internal verification: the gate accomplished what it should, but was nevertheless unable to distinguish the right rule due to the underdeterminacy of the puzzle.

6 Limitations

The 32 problems are those in the first 200 rows of the released LingOly-TOO test file. They were chosen by file order, not by system performance, thus the mix of problem types may not be representative of all 82 problems, though the split itself was fixed before any module was developed (Section 4). Consequently, the commit frontier detailed in Section 5.2 characterizes which families this system verifies on our selected subset, not provably generalizing to the full benchmark. Our results suggest that the modules are not able to carry over to new problem layouts—M1, M3,

M4, and M5 made zero commits across the 50 unseen problems, and every commit came from M2, at 35.6% precision. Therefore, we demonstrate a gate rather than a comprehensive routing system. Passing the gate is necessary for a correct solution, but it is not sufficient, as shown by Problems 139 and 91 (Section 5.3, Appendix A), which provided rules that reproduce every given but committed wrong answers. We suspect precision could be improved by re-fitting rules under leave-one-out, though the evidence consists of two problems so it is not yet a proven fix. Lastly, the majority of the aggregate exact-match lift of +3.65 points is within one problem (scrambled-numeral), and thus is not evidence of broad end-to-end gains (Section 5.2). We do not compare against confidence- or uncertainty-based abstention baselines; a head-to-head comparison is left to future work.

7 Conclusion and Future Work

Our system consists of a model (Gemini 2.5 Flash) proposing only a mechanical skeleton, specifically the segmentation or rule shape, while deterministic code builds, solves, and verifies the candidate against the puzzle-internal givens. Each cell is only committed if the gate passes, else it abstains and falls back to the baseline. Across the development problems, we report 131 commits of which 128 were exact, meaning a 97.7% precision and 10.5% coverage. On the 50 unseen problems, however, the system makes 73 commits at 35.6% precision. Notably, 46 of the 47 errors come from a single problem which passed a LOO check perfectly, a known failure mode similar to Problem 139. We conclude that the verification is exact when the solver is deterministic (M3, M5), but a perfect score on the givens doesn't ensure a model-assisted rule is correct when the givens are not able to single it out. Re-fitting the rule under leave-one-out alongside a system with the model acting as the extractor presents natural future work.

The current modules abstain on reduplication, multi-slot clitics, and other unaccounted phenomena outside the five classes; expanding the classes directly increases coverage. Separately, a limited memory-augmented iterative trial recovered 1 of 5 previously-abstained cells on Problem 155 with no additional false commits, while the forbid-list forces the model to diversify its proposals. Lastly, the modules could be packaged as verifier-gated tools for an LLM solving IOL-style problems.

Use of AI Assistants

AI (Claude Code) was used for LaTeX, grammar checks, sentence-level editing, tables, coding and running experiments, and drafting of some passages in the camera-ready revision; all text was reviewed by the authors, who are responsible for the research design, experiments, analysis, and all claims.

References

Andrew M. Bean, Simi Hellsten, Harry Mayne, Jabez Magomere, Ethan A. Chi, Ryan Chi, Scott A. Hale, and Hannah Rose Kirk. 2024. LingOly: A benchmark of olympiad-level linguistic reasoning puzzles in low-resource and extinct languages. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*.

Mukund Choudhary, KV Aditya Srivatsa, Gaurja Aeron, Antara Raaghavi Bhattacharya, Dang Khoa Dang Dinh, Ikhlasul Akmal Hanif, Daria Kotova, Ekaterina Kochmar, and Monojit Choudhury. 2025. UNVEILING: What makes linguistics olympiad puzzles tricky for LLMs? In *Proceedings of the Conference on Language Modeling (COLM)*.

Mathias Creutz and Krista Lagus. 2007. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)*, 4(1):3:1–3:34.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Satyam Goyal and Soham Dan. 2025. IOL-BENCH: Benchmarking LLMs on linguistic reasoning. ArXiv preprint arXiv:2501.04249.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics (TACL)*.

Katharina Kann, Arya D. McCarthy, Garrett Nicolai, and Mans Hulden. 2020. The SIGMORPHON 2020 shared task on unsupervised morphological paradigm completion. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 51–62.

Jude Khouja, Lingyi Yang, Karolina Korgul, Simi Hellsten, Vlad Neacsu, Harry Mayne, Ryan Kearns, Andrew Bean, and Adam Mahdi. 2026. LingOly-TOO: Disentangling reasoning from knowledge with templatised orthographic obfuscation. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ArXiv:2503.02972 (2025).

Da-Chen Lian, Ri-Sheng Huang, Pin-Er Chen, Chunki Lim, You-Kuan Lin, Guan-Yu Tseng, Zhen-Yu Lin, Pin-Cheng Chen, and Shu-Kai Hsieh. 2025. LOBSTER: Linguistics olympiad benchmark for structured evaluation on reasoning. In *Proceedings of the 37th Conference on Computational Linguistics and Speech Processing (ROCLING)*, pages 193–229.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the International Joint Conference on Natural Language Processing and the Asia-Pacific Chapter of the Association for Computational Linguistics (IJCNLP-AACL)*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Isidor Konrad Maier and Matthias Wolff. 2023. Arithmetics-based decomposition of numeral words — arithmetic conditions give the unpacking strategy. ArXiv preprint arXiv:2312.10097.

Neh Majmudar and Elena Filatova. 2025. Can LLMs generate and solve linguistic olympiad puzzles? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Neh Majmudar, Anne Huang, Jinfan Frank Hu, and Elena Filatova. 2026. From rosetta to match-up: A paired corpus of linguistic puzzles with human and LLM benchmarks. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*.

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

Arne Rubehn, Christoph Rzymski, Luca Ciucci, Katja Bocklage, Alžběta Kučerová, David Snee, Abishek Stephen, Kellen Parker van Dam, and Johann-Mattis List. 2025. Annotating and inferring compositional structures in numeral systems across languages. In *Proceedings of the 7th Workshop on Research in Computational Linguistic Typology and Multilingual NLP (SIGTYP 2025)*, pages 29–42.

Jon Saad-Falcon, E. Kelly Buchanan, Mayee F. Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Frederic Sala, Scott Linderman, Azalia Mirhoseini, and Christopher Ré. 2025. Weaver: Shrinking the generation-verification gap by scaling compute for verification. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Gözde Gül Şahin, Yova Kementchedjhieva, Phillip Rust, and Iryna Gurevych. 2020. PuzzLing Machines: A challenge on learning from small data. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics: EMNLP*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman. 2024. Hypothesis search: Inductive reasoning with language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Bingbing Wen, Jihan Yao, Shangbin Feng, Chenjun Xu, Yulia Tsvetkov, Bill Howe, and Lucy Lu Wang. 2025. Know your limits: A survey of abstention in large language models. *Transactions of the Association for Computational Linguistics (TACL)*, 13:529–556.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum.

2025. LiveBench: A challenging, contamination-limited LLM benchmark. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. ArXiv preprint arXiv:2311.04850.

Hanning Zhang, Shizhe Diao, Yong Lin, Yi R. Fung, Qing Lian, Xingyao Wang, Yangyi Chen, Heng Ji, and Tong Zhang. 2024. R-Tuning: Instructing large language models to say 'I don't know'. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

A Worked Example: Problem 139

Problem 139 is an M2 morphology problem with a plural-only noun. The model proposed an affix pattern, while the deterministic code induced the rule and applied the leave-one-out check. The induced rule was able to reproduce all 28 givens exactly, scoring 1.00 across the 26 testable leave-one-out folds; two of the givens were exempt under the atomic-witness rule, as the discriminating checks were satisfied (LOO reproduction = 1.0 and determinacy of ≥ 2 cases agreed). The gate committed `uingani` while the gold form is `engani`, thus the cell is incorrect. The issue lay in that the givens under-determined the rule, as they were consistent with multiple generalizations, and the queried cell existed where the generalizations diverge. The gate cannot detect this, only checking reproduction of the givens it has. Thus, a perfect verifier score is necessary but not sufficient, showing a ceiling of what puzzle-internal verification can guarantee.

B Forced Commit Ablation

We tested an ablation via disabling the determinacy guard and forcing the harness to commit on all 125 (30 cells from problem 167, 95 cells from problem 174) under-determined held-out cells, configuring it to emit a single constant value for every cell in a matching row. The gold value in each matching row is a permutation, consequently a constant matches exactly one cell per row. Across 10 matching rows, the permutation structure of each row forces 10 exact cells, consistent with the observed 8.0% precision: one exact cell per row is the analytic ceiling for any gold-blind constant policy, so 8.0% is the expected rate rather than a measured baseline.

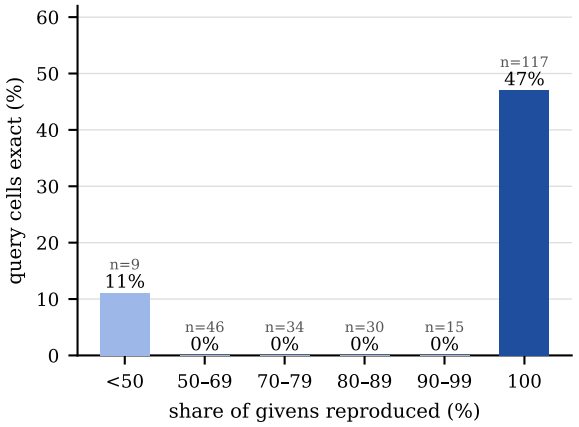

Figure 3: Exactness of the query-cell answers each proposed M2 hypothesis would have committed, bucketed by the share of given forms the hypothesis reproduces. Answers from sub-perfect hypotheses are unusable (0 of 125 exact at 50–99%); the single exact cell below 50% (1 of 9) is chance-level. Bar labels count query cells, not hypotheses.

C Hypothesis-Score Ablation

We reaccessed the frozen run's cached responses (no live model calls) and record all M2-proposed grammar hypotheses (45) across the tested rows. We also include rejected attempts, then score the query-cell answers each hypothesis would have produced, measuring the precision. Hypotheses reproducing 50–99% of the given forms answer 0 of 125 query cells correctly; hypotheses reproducing 100% answer 55 of 117 (47%), which the remaining abstention gate rules (leave-one-out, determinacy, coverage) raise up to 83% on M2's commits during our actual test run (Figure 3). Separately, we sweep the LOO commit threshold from 1.0 to 0.0 over the same frozen responses, gaining zero extra commits or committed answers as a result. This is because the reproduce-all-givens gate rejects every defective hypothesis before the LOO pass counts as verified. As the hypotheses reproducing 50–99% of the givens produce no acceptable answers, the low commit ratio is the cost of this cliff (Figure 3), not an adjustable conservatism.

D Second-Model Replication

We rerun the model-calling modules (M1, M2, M4) individually with a second model, Gemini 3.5 Flash, on the dev-split. Commits decreased

across the board: M1 commits on 0 of 10 numeral variants (2.5 Flash: 2 cells, both exact), M4 commits on 0 of 5 harmony variants (2.5 Flash: 1 cell, exact), and M2 commits on 2 variants across 6 problems, 5 cells in total (2.5 Flash: 8 variants, 18 cells). Both M2 commits passed the gate at a perfect verifier score (reproduce = LOO = 1.0): problem 135's 3 cells are all exact, while problem 16's 2 cells are both wrong. A wrong analysis thus again passed a perfect score, repeating the under-determinacy ceiling of Appendix A under a different model. No hypothesis-answer was committed without a perfect score on LOO under either model, so the gate maintained identical criteria throughout. The number of verifiable proposals hinges on the proposer, as our extraction process was dev-fitted using 2.5 Flash's responses, explaining the commit gap. This explanation becomes evident after considering 3.5 Flash's freeform accuracy, which is equal or higher than 2.5 Flash's on seven of the eight probed problems (0.84 vs 0.20 on problem 155), thus the reduction is attributable to poor extraction fit. M3 and M5 are zero-model-call modules with deterministic outputs and thus were excluded from this re-run.