LLMEval-Logic: A Solver-Verified Chinese Benchmark for Logical Reasoning of LLMs with Adversarial Hardening

Anonymous ACL submission

Abstract

Evaluating large language models (LLMs) on natural-language logical reasoning is essential because rule-governed tasks require conclusions to follow strictly from stated premises. Many existing logical-reasoning benchmarks are generated by templating natural-language items from sampled formulas, provide only coarse or unaudited formal annotations, and are now quickly saturated by frontier reasoning models. We present LLMEval-Logic, a Chinese logical reasoning benchmark built from realistic situational scenarios. Its pipeline forward-authors and expert-audits natural-language items together with their reference formalizations, verifies annotated answers with Z3, constructs expert rubrics for natural-to-formal grading, and hardens selected items through a closed-loop adversarial workflow. The benchmark is released in two paired subsets: a 246-item Base subset shipped with 1,400 expert-developed rubric atoms, and a 190-item Hard subset with 938 multi-step subquestions over closed model spaces. Evaluating 14 frontier LLMs on LLMEval-Logic reveals substantial gaps in current models: the best model reaches only 37.5% Hard Item Accuracy, and even with reference symbols the highest joint Z3+Rubric formalization score among evaluated models reaches only 60.16%.

Title: Music Composition

BACKGROUND. Considering harmonic fit, when composing this passage, the composer only needs some sections of the orchestra to cooperate. The orchestra currently has four candidate sections to add: trumpet, horn, alto horn, and tuba. The rules for harmonic comfort are as follows:

1. If the horn is added, the tuba must be added.
2. The alto horn is added if and only if the trumpet is added and the horn is not added.
3. If the tuba is added, then the alto horn is not added.
4. Either the trumpet is added and the tuba is not added, or the tuba is added and the trumpet is not added.
5. The trumpet and the alto horn cannot both be added.
6. The trumpet and the horn cannot both be absent.

QUESTION. Which combinations of sections can ultimately be added?

REFERENCE ANSWER. Horn and tuba.

Figure 1: A representative Base item (English translation): the model enumerates feasible solutions for the closed-world scenario, scored against a Z3-verified reference. Hard items chain ∼5 counterfactual subquestions (Appendix G).

1 Introduction

Logical reasoning over natural language underpins many high-stakes applications of LLMs, from contract review and regulatory compliance to clinical guideline checking and automated grading of multi-step proofs, where conclusions must follow strictly from stated premises rather than from surface pattern matching (Parmar et al., 2024; Han et al., 2024a). Reliable benchmarking of this capability is therefore a prerequisite for trustworthy deployment of LLMs in rule-governed tasks. Current logical reasoning benchmarks, however, fall short along three complementary dimensions, and we organize our motivation around the following challenges.

Challenge 1: How can items remain semantically grounded in realistic scenarios? A large body of work generates items by reverse construction, where a formal structure is sampled first and then rendered into natural language (Clark et al., 2020; Tafjord et al., 2021; Parmar et al., 2024; Wei et al., 2025; Nadas et al., 2025). Such items inherit template-like phrasing and formula-shaped discourse that drift from realistic scenarios, leaving distributional artifacts that allow LLMs to predict the answer from surface cues without performing the underlying inference (Wu et al., 2024; Jiang et al., 2024; Xie et al., 2025).

Challenge 2: How can natural-to-formal translation be audited at the level of local semantic units, not just final answers? Human-written logical narratives (Han et al., 2024a,b) improve

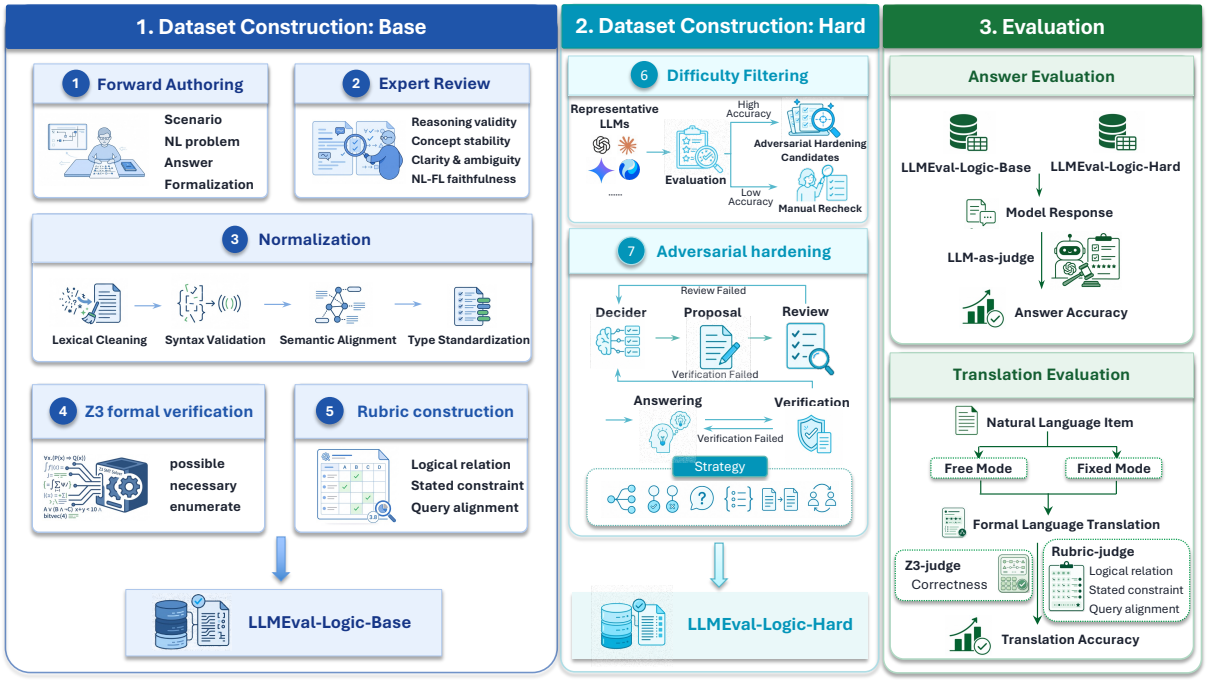


Figure 2: Overview of LLMEval-Logic. The construction pipeline forward-authors items from realistic scenarios, audits the gold natural-language → formal-language alignment, verifies annotated answers with Z3, constructs expert rubrics, and hardens selected items through a closed-loop adversarial agent workflow. The released benchmark exposes paired Base and Hard subsets, evaluated along two complementary axes: *answer accuracy* (Item / Sub-Q) for model responses, and *formalization accuracy* (Z3 execution + rubric matching) for model-produced formalizations.

naturalness but release only limited or unaudited formal annotations and provide no rubric for scoring model-produced formalizations. As a result, even when a model emits a candidate FOL translation, existing benchmarks cannot tell whether logical relations, stated constraints, and the query are faithfully encoded (McIntosh et al., 2026).

Challenge 3: How can a benchmark stay hard enough to discriminate among frontier reasoning models? Many published splits saturate quickly under modern reasoning models, with top systems clustering near the ceiling on item-level accuracy and leaving little headroom to localize where these models still fail (Suzgun et al., 2023; Kazemi et al., 2025).

A theoretical perspective. Automated reasoning faces ultimate complexity limits: propositional satisfiability (SAT) is NP-hard (Cook, 1971), and first-order logic (FOL) satisfiability is undecidable (Turing, 1937; Church, 1936). These bounds are insurmountable for any formal reasoning system, including SMT solvers and proof assistants. LLMs, however, operate in natural language and produce answers regardless of proof complexity (Saparov and He, 2023). For them, a more immediate empirical bottleneck appears well before any computational complexity wall: LLMs struggle to identify the logical relations, types, and quantifiers needed to faithfully formalize natural-language reasoning (Yang et al., 2024; Xu et al., 2025; Cao et al., 2026). Robust logical alignment, namely faithful natural-to-formal translation, therefore remains a critical open challenge that motivates LLMEval-Logic.

To address these challenges, we present **LLMEval-Logic**, a Chinese logical reasoning benchmark constructed through a three-stage audited pipeline. *First*, trained authors with prior coursework in logic forward-author each item from a real situational scenario, such as eligibility rules, scheduling, roles, or institutional procedures, and pair it with a formal-language formalization. The formalization is then reviewed by annotators with graduate-level training in formal-reasoning disciplines. *Second*, every item passes a four-layer normalization pipeline that covers lexical, syntactic, semantic, and type-and-parameter checks. The normalized item then enters a Z3 verification stage, where the solver checks whether the formalized premises and queries support the annotated answer under the corresponding SAT, entailment, or model-enumeration semantics, dispatched by one of three base task labels (`possible`, `necessary`,

	Base	Hard
# Items	246	190
# Sub-questions	246	938
Avg. sub-Qs / item	1	4.94
# Question types	3	6

Table 1: Composition of LLMEval-Logic. Base covers single-question PL / FOL reasoning and formalization grading, with 1,400 rubric atoms (756 logical-relation, 398 stated-constraint, 246 query-alignment). Hard contains adversarially-hardened multi-question items; an item is correct only if every sub-question is correct.

`enumerate_models`). The underlying SAT, entailment, and model-enumeration semantics are made precise in Section 2.1. Each verified item further carries expert-developed rubrics that decompose natural-to-formal faithfulness into logical relations, stated constraints, and query alignment. *Third*, items deemed too easy after model probing enter a closed-loop adversarial hardening workflow with five agent roles: Decider, Proposal, Review, Answering, and Verification. The workflow rewrites them along six hardening strategies until each accepted Hard item passes review and verification with a per-step audit trace.

The resulting benchmark is released in two paired subsets: Base for audited single-question reasoning and natural-to-formal evaluation, and Hard for multi-question adversarial reasoning over the same forward-authored item base.

Our contributions are threefold:

1. We release *LLMEval-Logic*, a forward-authored, Z3-audited Chinese logical reasoning benchmark with expert-developed rubrics and paired Base / Hard subsets.
2. We design a closed-loop adversarial hardening workflow (Decider, Proposal, Review, Answering, Verification) that rewrites high-accuracy items along six structural strategies and ships only items that pass review and verification.
3. We conduct a systematic evaluation of 14 frontier LLM variants over three independent runs, assessing both natural-language logical reasoning and faithful natural-to-formal translation.

2 Design

2.1 Dataset Construction

LLMEval-Logic is built around three construction objectives: to author Chinese reasoning items through forward authoring rather than reverse-templating, to pair each item with a Z3-verified formal representation and expert-developed rubrics, and to calibrate benchmark difficulty so that items remain challenging for frontier LLMs. Base-item construction is organized into five sequential audit stages (forward authoring, expert review, layered normalization, Z3-based formal verification, and rubric construction); validated Base items then enter the adversarial hardening pipeline of Section 2.2 to produce the Hard subset. Table 1 summarizes the released composition and Figure 3 visualizes its distribution along question type and Hard sub-question count.

Forward Authoring, Expert Review, and Normalization. LLMEval-Logic starts from 521 human-seeded Chinese reasoning items written by contributors with prior coursework in logic, who embed formalizable logical structures inside naturalistic scenarios (rules, schedules, eligibility, institutional procedures) rather than back-translating sampled formulas. Annotators with graduate-level training in formal-reasoning disciplines then review concept stability, reasoning validity, clarity, and NL–FL faithfulness, tracing every variable, predicate, premise, and labelled answer back to the Chinese problem statement (Lean is used as an auxiliary sanity check on selected items). Reviewed items pass through a four-layer normalization pipeline (lexical, syntactic, semantic-alignment, type/parameter) that turns the human-audited formalization into a consistent, parseable, and auditable object without changing its semantics; layer-by-layer normalization details are in Appendix A.

Formal Verification. We employ the Z3 SMT solver to perform formal verification on the normalized and human-audited formalizations, dispatching both the formalized premises and the formalized query to the solver. This stage serves a dual purpose: it verifies that the labelled answer is logically certified by the appropriate solver operation, and it surfaces residual formalization flaws such as missing or redundant premises, question-type mismatches, or inconsistent symbol mappings.

We categorize Base questions into three task types — `possible`, `necessary`, and `enumerate_models` — treated as NLP task labels mapped to distinct Z3 solving modes rather than as strict modal-logic operators. For a premise set Σ and a query φ, `possible` checks satisfiability $\textsc{Sat}(\Sigma \cup \{\varphi\})$ (equivalent to $\Sigma \not\models \neg\varphi$); `necessary`

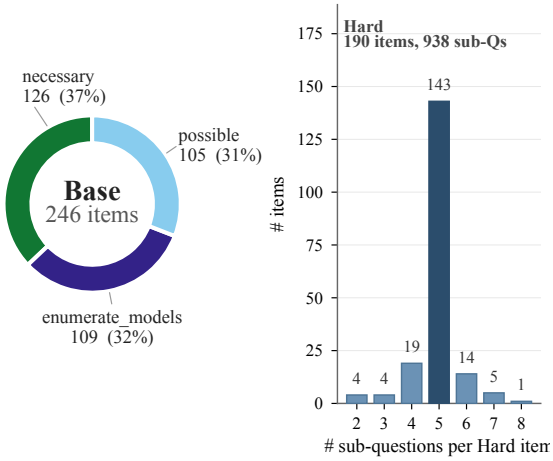


Figure 3: Composition of the two released subsets. Left: Base question-type distribution (items may carry multiple labels) and PL / FOL split. Right: Hard sub-question count distribution (highlighted bar = mode at 5).

checks entailment $\Sigma \vDash \varphi$ via $\textsc{Unsat}(\Sigma \cup \{\neg\varphi\})$; and `enumerate_models` returns the distinct satisfying assignments over the closed scenario. Failed checks trigger item repair and Z3 re-verification, so that the clean benchmark contains only items that pass formal verification.

Expert-Developed Rubrics. Formal verification certifies that the labelled answer follows from the normalized reference formalization, but it does not by itself provide a fine-grained criterion for evaluating model-produced formalizations. We therefore construct expert-developed rubrics for each Base item to audit NL–FL faithfulness at the level of local semantic units. The rubrics decompose the reference formalization into three groups: `logical_relation` checks whether core relations such as implication, negation, disjunction, exclusivity, and quantification are encoded correctly; `stated_constraint` checks whether explicit facts, boundary conditions, object domains, and type restrictions are preserved; and `query_alignment` checks whether the formal query matches the natural-language question. Each rubric is stored in two aligned forms: a natural-language criterion for semantic inspection and a Z3-checkable statement for automatic scoring. We calibrate all rubrics on the expert-confirmed reference formalization, requiring the gold formalization to pass the full natural-language rubric set before downstream NL-to-FL evaluation.

LLMEval-Logic-Base. LLMEval-Logic-Base is the single-question subset used for answer evaluation and formalization grading. Every item passes expert review, normalization, and Z3-based formal verification, and uses one or more of the three base question types defined above: `possible`, `necessary`, and `enumerate_models`. Each Base item is paired with expert-developed rubrics for grading model-produced formalizations.

2.2 Adversarial Hardening

Hardening Workflow. After expert review, normalization, and verification, candidate items are probed with representative LLMs to estimate their empirical difficulty. The resulting probe scores are used only for routing, never for answer validation: low-accuracy items are manually rechecked and retained in Base if validated, while high-accuracy items enter a closed-loop hardening workflow with five agent roles. The five roles implement this hardening loop as follows. **Decider** diagnoses the shallow solution path and emits a hardening blueprint, **Proposal** rewrites the background and questions, **Review** checks well-posedness and candidate-space closure, **Answering** produces independent solutions to expose residual ambiguity, and **Verification** flags unsupported conclusions or missing recomputation under counterfactual variants. The blueprint draws on six hardening strategies: branching, valid distractors, explicit uncertainty, set-valued outputs, counterfactual variants, and alias/coreference variation. These strategies are chosen to alter the closed candidate space, change the required operation, or suppress lexical shortcuts rather than reword the surface. Failed review or verification loops back to proposal; answer-only failures trigger answer repair. Only items that pass both gates are included in LLMEval-Logic-Hard; detailed agent configuration, retry budgets, and gate-convergence statistics are in Appendix B.

LLMEval-Logic-Hard. LLMEval-Logic-Hard contains 190 hardened items totaling 938 sub-questions, with an average of 4.94 sub-questions per item (range 2–8). The hardening workflow extends the Base type schema with two closed-world operations, `unique_solution` (whether the constraints determine exactly one feasible assignment) and `has_alternative` (whether another feasible assignment exists under added conditions), so that an item can probe whether a model truly searches the entire model space rather than terminating on the first feasible candidate. Together with multi-subquestion formatting, these

Model	Base	Hard	
	Item Acc. (%)	Item Acc. (%)	Sub-Q Acc. (%)
Thinking models			
Gemini 3.1 Pro (2026)	74.0 ± 1.2	$\mathbf{37.5 \pm 3.8}$	76.0 ± 0.6
Claude Opus 4.6 (2026)	68.7 ± 3.2	36.7 ± 2.6	$\mathbf{76.6 \pm 1.4}$
GPT-5.4 Pro (2026b)	71.8 ± 0.9	32.6 ± 4.3	68.4 ± 3.1
Qwen 3.5 Plus (2026)	71.3 ± 0.6	29.3 ± 4.0	70.0 ± 1.9
Kimi K2.5 (2026)	72.9 ± 3.0	28.1 ± 2.4	68.4 ± 1.1
Hy3 preview (2026)	75.3 ± 3.1	21.6 ± 0.9	62.7 ± 0.7
Seed 2.0 Pro (2026)	$\mathbf{75.5 \pm 1.3}$	20.4 ± 5.4	63.3 ± 2.0
No-think / low-think models			
Claude Opus 4.6 (no-think) (2026)	69.0 ± 1.2	36.0 ± 2.4	74.4 ± 1.2
Gemini 3.1 Pro (low-think) (2026)	73.3 ± 0.9	33.5 ± 2.6	71.3 ± 1.0
GPT-5.4 Pro (no-think) (2026b)	71.8 ± 0.9	33.0 ± 2.0	70.0 ± 2.1
Seed 2.0 Pro (no-think) (2026)	56.2 ± 2.8	6.1 ± 0.3	39.4 ± 0.9
Qwen 3.5 Plus (no-think) (2026)	42.0 ± 0.2	3.0 ± 0.3	33.7 ± 0.8
Kimi K2.5 (no-think) (2026)	37.9 ± 0.9	1.8 ± 1.1	27.0 ± 0.6
Hy3 preview (no-think) (2026)	51.4 ± 1.0	0.9 ± 1.1	27.4 ± 0.5

Table 2: LLMEval-Logic leaderboard on Base (single-question Item Acc.) and Hard (Item Acc. plus Sub-Q Acc.), `gpt-5.1-chat` judge, mean $\pm$ std over 3 runs. Inter-judge agreement against two further frontier judges (Claude Opus 4.6, Gemini 3.1 Pro) is reported in Appendix D. Bars encode the score on a 0–100 scale. Hard Item Acc. requires all sub-questions of an item to match; Sub-Q Acc. scores individual sub-questions. Rows sorted by Hard Item Acc.; best per column in bold.

operations require models to maintain and update the full candidate space across related queries.

2.3 Evaluation Protocol

LLM-as-Judge and Metrics. All responses are scored semantically by an LLM-as-Judge using `gpt-5.1-chat` (OpenAI, 2026a), used both for the answer-side judge that produces Item Accuracy / Sub-Q Accuracy and for the rubric-side judge that scores each rubric atom. Base reports Item Accuracy over single-question items. Hard reports both Item Accuracy, where every sub-question in an item must match, and Sub-Question Accuracy, which aggregates over individual sub-questions. We re-judge two random samples (103 sub-questions, 105 rubric atoms) with two additional frontier judges, Claude Opus 4.6 and Gemini 3.1 Pro, under the same prompts; pairwise Cohen's $\kappa \in [0.873, 0.922]$ throughout and all three judges return identical verdicts on 93% of items, comfortably above the *almost perfect* threshold of Landis and Koch (1977); full numbers and protocol are in Appendix D, with answer-evaluation details in Appendix C. To control for evaluation noise we run each model three times under independent sampling and report all numbers as mean and sample standard deviation.

Formalization Evaluation. Beyond answer accuracy, each Base item is also scored at the formalization level under two settings: *free*, where the model chooses its own symbols and translations, and *fixed*, where the reference symbol inventory and glosses are provided. Two complementary signals are computed (**Z3** = solver execution against the reference; **Rubric** = pass rate over logical-relation, stated-constraint, and query-alignment atoms; **Both** requires the two to agree on the same item), with protocol details in Appendix C.

3 Experiment and Analysis

We organize the analysis around three research questions, each addressing one challenge from Section 1: **RQ1** (Challenge 1) asks whether forward-authored, audited items resist template-style short-

cuts; **RQ2** (Challenge 2) asks how faithfully frontier LLMs translate natural-language items into formal logic when audited at the level of local semantic units; **RQ3** (Challenge 3) asks whether LLMEval-Logic-Hard provides enough discriminative headroom on frontier reasoning models and where their failures localize.

3.1 Experimental Setup

We evaluate 14 frontier LLMs across 7 model families and two configurations (thinking, no-thinking) over three independent runs under the protocol of Section 2.3, reporting mean $\pm$ std. Table 2 reports the consolidated answer-level results, and Table 3 reports the formalization-side results on the 246 Base items, including the gold reference formalization as an upper-bound calibration that confirms the reference passes both Z3 execution and rubric matching.

3.2 RQ1: Shortcut Resistance of Forward-Authored Items

Finding 1: Forward-authored Hard items remain difficult. Across the 14 evaluated models, Item Accuracy averages 65.1% on Base but only 22.9% on Hard, a 42.2-point drop; the strongest Hard score is just $37.5 \pm 3.8\%$ (Table 2, Figure 5a). The forward-authored construction keeps the language and scenarios natural rather than template-like; under this realistic setting, controlled hardening still drives Hard accuracy down sharply through closed-world recomputation, branching, distractors, set-valued outputs, uncertainty, and alias/coreference tracking (Appendix F). Moreover, the low Hard accuracy reflects persistent limitations of current models in complex natural-language logical reasoning.

3.3 RQ2: Faithfulness of Natural-to-Formal Translation

Finding 2: Z3 and rubric signals are complementary for faithful formalization. For every evaluated model, the joint Z3+Rubric score is lower than either individual signal in both free-symbol and fixed-symbol settings (Table 3). Z3 execution can accept candidate FLs that reach the reference answer while missing or distorting source semantics, especially in low-cardinality answer spaces; rubric scoring checks required semantic atoms but, as a positive checklist, cannot rule out all extra or mis-specified constraints. These complementary

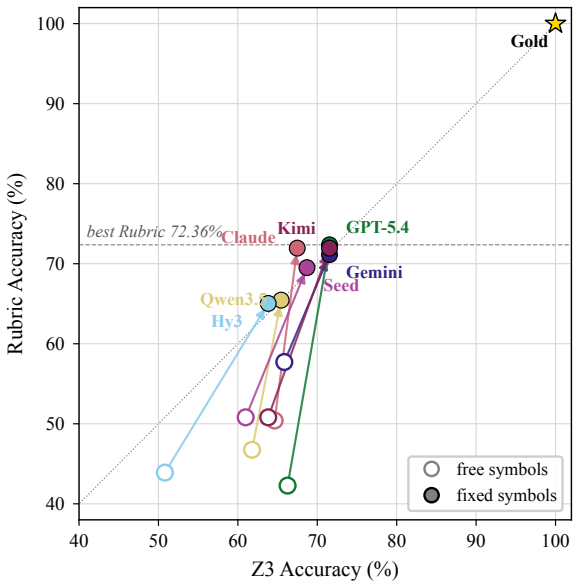


Figure 4: Z3 vs Rubric accuracy on Base (thinking variants; full data in Table 3). Hollow=free symbols, filled=fixed; arrows go free→fixed. Most points lie below $y = x$ (Finding 2); fixed shifts arrows up-right but no model crosses the 72.36% Rubric ceiling or reaches the gold star at $(100, 100)$ (Finding 3).

blind spots make the joint metric a stricter practical proxy for NL-to-FL faithfulness (Appendix E).

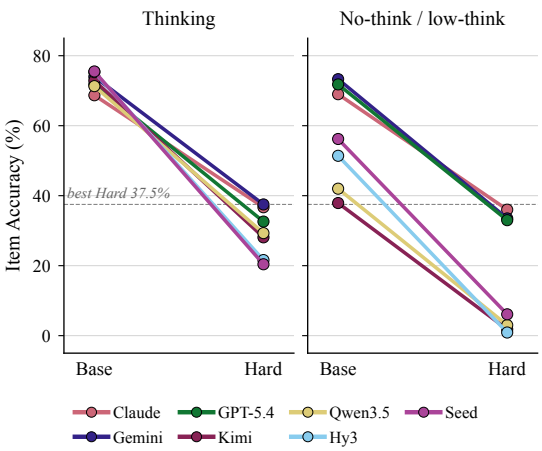
Finding 3: Fixed symbols help but do not solve translation. Providing the reference symbol inventory and glosses moves nearly all models upright in Figure 4, showing that symbol choice and symbol grounding are real sources of difficulty. The gains are substantial for weaker configurations, such as GPT-5.4 Pro (no-think) rising from 32.93% to 60.16% Z3+Rubric. Yet the best non-gold Fixed Z3+Rubric score remains 60.16%, far below the 100% gold reference. The residual gap shows that even when the correct symbols are supplied, models still fail to encode premises and queries with the semantic strength required by the item.

3.4 RQ3: Discriminative Headroom on Frontier Reasoning Models

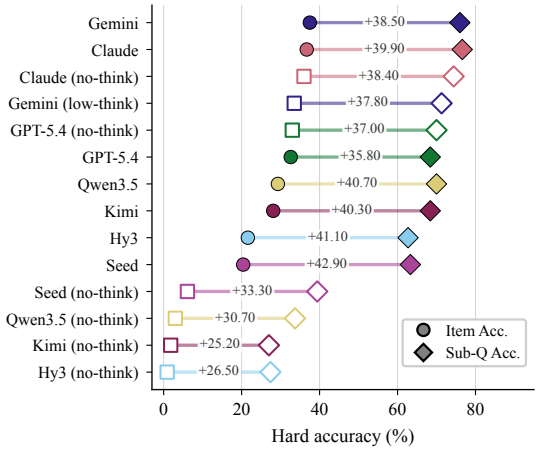
Finding 4: Hard sharply lowers accuracy and exposes family-specific collapse. Figure 5a shows that every model family drops from Base to Hard, revealing that even frontier models struggle in adversarially hardened, multi-question closed-world scenarios that require maintaining the full candidate space across related queries. The configuration effect is family-specific: open-weight Chinese models (Hy3 preview, Qwen 3.5 Plus, Kimi K2.5) lose 20–26 pp of Hard Item Accuracy when think-

Model	Free symbols			Fixed symbols		
	Z3	Rubric	Both	Z3	Rubric	Both
Thinking models						
Gemini 3.1 Pro	65.85	**57.72**	**45.12**	**71.54**	71.14	58.13
Claude Opus 4.6	64.63	50.41	38.62	67.48	71.95	56.91
GPT-5.4 Pro	**66.26**	42.28	30.08	**71.54**	**72.36**	58.94
Qwen 3.5 Plus	61.79	46.75	36.18	65.45	65.45	52.44
Kimi K2.5	63.82	50.81	36.18	**71.54**	71.95	**59.35**
Seed 2.0 Pro	60.98	50.81	35.77	68.70	69.51	56.91
Hy3 preview	50.81	43.90	27.64	63.82	65.04	51.22
No-think / low-think models						
GPT-5.4 Pro (no-think)	**69.11**	42.68	32.93	**72.36**	**72.36**	**60.16**
Claude Opus 4.6 (no-think)	66.26	**56.50**	43.50	60.57	64.63	50.00
Gemini 3.1 Pro (low-think)	66.26	55.28	**44.31**	69.92	67.48	54.47
Seed 2.0 Pro (no-think)	49.19	47.56	30.89	60.16	58.54	47.56
Qwen 3.5 Plus (no-think)	56.10	35.37	25.20	55.28	45.12	35.77
Hy3 preview (no-think)	45.93	34.96	21.95	41.87	40.24	24.80
Kimi K2.5 (no-think)	44.72	37.80	27.24	52.85	47.97	38.62
Gold reference	100.00	100.00	100.00	100.00	100.00	100.00

Table 3: Formalization accuracy on Base (%), Z3 and Rubric judges both running on `gpt-5.1-chat` (validated against two further frontier judges in Appendix D). Z3: solver execution against the reference; Rubric: pass rate over logical-relation, stated-constraint, and query-alignment atoms; Both: the two signals agree on the same item. Rows are split into thinking and no-think/low-think blocks and sorted by Free-symbols Both within each block. Shading is column-normalized; bold marks the best non-gold per column.



(a) Base→Hard Item Accuracy per family (thinking left, no-thinking right).



(b) Hard Item Acc. (circles) vs Sub-Q Acc. (diamonds); filled = thinking, hollow = no-thinking; in-line number = Sub-Q−Item gap.

Figure 5: RQ3 visuals across 14 frontier LLM variants from 7 families: per-family Base→Hard Item Accuracy in (a), and per-variant Hard Item vs Sub-Q Accuracy with the in-line Sub-Q−Item gap in (b). Panel (a) makes the Base→Hard drop universally visible and the no-thinking effect family-specific; panel (b) localises the remaining headroom at item-level aggregation. See Findings 4–5 (numbers in Table 2, diagnostics in Appendix F).

ing is disabled, while proprietary frontier models (Claude Opus 4.6, GPT-5.4 Pro) remain flat or improve slightly. At the same time, Hard produces wider performance gaps across model families than Base: different model families show larger capability differences when reasoning over a complex closed candidate space.

Finding 5: Base and Hard rankings diverge for frontier configurations. Re-ranking the seven thinking variants by Base and Hard Item Accuracy (Figure 6) shows a clear inversion: Seed 2.0 Pro and Hy3 preview rank at the top on Base (75.5%, 75.3%) but fall near the bottom on Hard (20.4%, 21.6%), whereas Claude Opus 4.6 rises

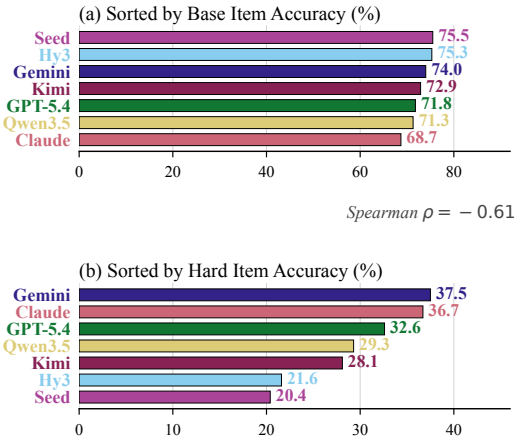


Figure 6: Thinking-variant ordering by Base Item Accuracy (top) and by Hard Item Accuracy (bottom). The same family colour appears in both panels, so the rank inversion is visible as a re-shuffling of the colour sequence (thinking $\rho = -0.61$; no/low-thinking $\rho = +0.79$, Table 2).

from the lowest Base rank to Hard rank 2 (Spearman $\rho = -0.61$). This divergence shows that high single-question accuracy does not reliably transfer to adversarial multi-question reasoning over closed candidate spaces. The no/low-thinking variants show a positive correlation because Chinese open-weight families drop on both Base and Hard, so the ranking inversion is most pronounced among thinking-enabled frontier configurations.

Finding 6: Hard difficulty comes from closed-space maintenance across chained queries. The 25–43-point Sub-Q→Item gaps in Figure 5b show that many models can solve individual sub-questions but fail the all-subquestions-correct item criterion. Appendix F explains why: Hard operators force global recomputation under counterfactual edits and exhaustive maintenance of enlarged candidate spaces under branching, distractors, uncertainty, set-valued outputs, and coreference across related sub-questions. Case studies in Appendix G instantiate these mechanisms across recurrent Hard failures: models locally patch counterfactual edits without recomputing dependent branches, under-enumerate interacting uncertainty axes, confuse element-, set-, and set-family-level projections, and lose evidence-provenance or validity-tier boundaries.

4 Related Work

Logic benchmarks. Many logical reasoning benchmarks generate items from symbolic rule worlds (RuleTaker (Clark et al., 2020), ProofWriter (Tafjord et al., 2021), LogicBench (Parmar et al., 2024), SATBench (Wei et al., 2025)); FOLIO (Han et al., 2024a,b) adds human-written narratives with FOL annotations. These resources have advanced natural-language reasoning evaluation, but formula-derived construction can leave template-like artifacts, and final-answer scoring alone cannot audit whether a model preserves the source semantics in formalization. LLMEval-Logic builds on the LLMEval evaluation framework (Zhang et al., 2024, 2023, 2025b,a) by contributing forward-authored Chinese scenarios paired with expert-audited reference formalizations and Z3-verified answers.

NL-to-FL eval and hardening. A complementary line studies NL-to-FL translation (Yang et al., 2024; Lee et al., 2025; Pei et al., 2025); but answer- or execution-level signals do not localize which part of a candidate formalization mis-encodes the source. We therefore score model-produced formalizations with expert rubrics over logical-relation, stated-constraint, and query-alignment atoms. Other work raises difficulty via proof depth (Saparov and He, 2023; Saparov et al., 2023), SAT puzzles (Wei et al., 2025), counterintuitive compositions (Chung et al., 2025), and logic grids (Lin et al., 2025). LLMEval-Logic combines rubric-graded formalization evaluation with controlled hardening that restructures closed candidate spaces, yielding items that remain interpretable to humans, executable by solvers, and discriminative for current frontier LLMs.

5 Conclusion

We present LLMEval-Logic, a Chinese logical reasoning benchmark pairing Z3-verified Base items with adversarially-hardened Hard items, scored on answer accuracy and rubric-graded formalization faithfulness. Across 14 frontier LLMs and three runs, top systems reach only 37.5% Hard Item Accuracy, no-thinking collapse is family-specific, Base and Hard rankings diverge, and Z3 correctness overestimates rubric faithfulness; these signals show that Base accuracy, Hard accuracy, and formalization faithfulness capture distinct aspects of frontier logical reasoning. We will release the benchmark, workflow, and audit traces upon publication: forward authoring and Z3/rubric auditing form the quality-control foundation, while controlled hardening keeps logical reasoning evaluation discriminative as raw accuracy saturates.

Limitations

Despite the audited construction pipeline, LLMEval-Logic has several limitations. First, the benchmark currently focuses on Chinese natural-language items. The underlying formalizations are language-agnostic, but generalization of the natural-to-formal evaluation to other languages and writing conventions remains to be empirically validated. Second, our scope covers propositional logic and first-order logic. Higher-order logic, modal and temporal logic, and probabilistic logical reasoning are left to future extensions of the same pipeline. Third, formal verification with Z3 certifies that the labelled answer follows from the normalized reference formalization, but cannot by itself certify that the formalization is fully equivalent to the natural-language item. The expert-developed rubrics are designed to mitigate this gap, yet rubric construction depends on expert annotation effort and may admit residual blind spots at extremely fine-grained semantic units.

References

Anthropic. 2026. Claude Opus 4.6 System Card. Official system-card page for Claude Opus 4.6. Accessed 2026-05-11.

ByteDance Seed Team. 2026. Seed2.0 Model Card. Official Seed2.0 model page and model card, including Seed2.0 Pro. Accessed 2026-05-11.

Bowen Cao, Dongdong Zhang, Yixia Li, Junpeng Liu, Shijue Huang, Chufan Shi, Hongyuan Lu, Yaokang Wu, Guanhua Chen, Wai Lam, and Furu Wei. 2026. From abstract to contextual: What llms still cannot do in mathematics. *CoRR*, abs/2601.23048.

Tsz Ting Chung, Lemao Liu, Mo Yu, and Dit-Yan Yeung. 2025. DivLogicEval: A framework for benchmarking logical reasoning evaluation in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 901–915, Suzhou, China. Association for Computational Linguistics.

Alonzo Church. 1936. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3882–3890.

Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

Google DeepMind. 2026. Gemini 3.1 Pro Model Card. Official model card, published February 19, 2026. Accessed 2026-05-11.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024a. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031. Association for Computational Linguistics.

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

Bowen Jiang, Yangxinyu Xie, Zhuoqun Hao, Xiaomeng Wang, Tanwi Mallick, Weijie Su, Camillo J. Taylor, and Dan Roth. 2024. A peek into token bias: Large language models are not yet genuine reasoners. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 4722–4756. Association for Computational Linguistics.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. 2025. Big-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 26473–26501. Association for Computational Linguistics.

Kimi Team. 2026. Kimi K2.5: Visual Agentic Intelligence. *Preprint*, arXiv:2602.02276.

J. Richard Landis and Gary G. Koch. 1977. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174.

Jinu Lee, Qi Liu, Runzhi Ma, Vincent Han, Ziqi Wang, Heng Ji, and Julia Hockenmaier. 2025. Entailment-preserving first-order logic representations in natural language entailment. In *Proceedings of the 63rd Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), pages 5729–5742, Vienna, Austria. Association for Computational Linguistics.

Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. ZebraLogic: On the scaling limits of LLMs for logical reasoning. *arXiv preprint arXiv:2502.01100*.

Timothy R. McIntosh, Teo Susnjak, Nalin A. G. Arachchilage, Tong Liu, Dan Xu, Paul A. Watters, and Malka N. Halgamuge. 2026. Inadequacies of large language model benchmarks in the era of generative artificial intelligence. *IEEE Trans. Artif. Intell.*, 7(1):22–39.

Mihai Nadas, Laura Diosan, and Andreea Tomescu. 2025. Synthetic data generation using large language models: Advances in text and code. *IEEE Access*, 13:134615–134633.

OpenAI. 2026a. GPT-5.1 Chat Model. API model documentation for gpt-5.1-chat-latest. Accessed 2026-05-11.

OpenAI. 2026b. Introducing GPT-5.4. Official product announcement for GPT-5.4 and GPT-5.4 Pro, published March 5, 2026. Accessed 2026-05-11.

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

Yu Pei, Yongping Du, and Xingnan Jin. 2025. FoVer: First-order logic verification for natural language reasoning. *Transactions of the Association for Computational Linguistics*, 13:1340–1359.

Qwen Team. 2026. Qwen3.5: Towards Native Multimodal Agents. Official Qwen3.5 release note; includes Qwen3.5-Plus. Accessed 2026-05-11.

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

Abulhair Saparov, Richard Yuanzhe Pang, Vishakh Padmakumar, Nitish Joshi, Mehran Kazemi, Najoung Kim, and He He. 2023. Testing the general deductive reasoning capacity of large language models using OOD examples. In *Advances in Neural Information Processing Systems*, volume 36, pages 3083–3105.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

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

Tencent Hy Team. 2026. Hy3 preview. Official open-source repository and model card for Hy3 preview, released April 23, 2026. Accessed 2026-05-11.

A. M. Turing. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265.

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

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 1819–1862. Association for Computational Linguistics.

Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. 2025. On memorization of large language models in logical reasoning. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP-AACL 2025, Mumbai, India, December 20-24, 2025*, pages 2742–2785. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.

Ruiyang Xu, Jialun Cao, Yaojie Lu, Ming Wen, Hongyu Lin, Xianpei Han, Ben He, Shing-Chi Cheung, and Le Sun. 2025. CRUXEVAL-X: A benchmark for multilingual code reasoning, understanding and execution. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 23762–23779. Association for Computational Linguistics.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2024. Harnessing the power of

large language models for natural language to first-order logic translation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6942–6959, Bangkok, Thailand. Association for Computational Linguistics.

Ming Zhang, Yujiong Shen, Jingyi Deng, Yuhui Wang, Huayu Sha, Kexin Tan, Qiyuan Peng, Yue Zhang, Junzhe Wang, Shichun Liu, Yueyuan Huang, Jingqi Tong, Changhao Jiang, Yilong Wu, Zhihao Zhang, Mingqi Wu, Mingxu Chai, Zhiheng Xi, Shihan Dou, and 3 others. 2025a. Llmeval-fair: A large-scale longitudinal study on robust and fair evaluation of large language models. *Preprint*, arXiv:2508.05452.

Ming Zhang, Yujiong Shen, Zelin Li, Huayu Sha, Binze Hu, Yuhui Wang, Chenhao Huang, Shichun Liu, Jingqi Tong, Changhao Jiang, Mingxu Chai, Zhiheng Xi, Shihan Dou, Tao Gui, Qi Zhang, and Xuanjing Huang. 2025b. LLMEval-Med: A real-world clinical benchmark for medical LLMs with physician validation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 4888–4914, Suzhou, China. Association for Computational Linguistics.

Ming Zhang, Yue Zhang, Shichun Liu, Haipeng Yuan, Junzhe Wang, Yurui Dong, Jingyi Deng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Llmeval-2.

Yue Zhang, Ming Zhang, Haipeng Yuan, Shichun Liu, Yongyao Shi, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Llmeval: A preliminary study on how to evaluate large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19615–19622.

A Layered Normalization Pipeline

Raw items submitted by different authors vary in symbol style, variable naming, LaTeX formatting, PL/FOL typing, and parameter conventions. Before formal verification, every item passes a four-layer normalization pipeline, decoupled so that any single layer can be re-run on a single item without invalidating downstream artifacts:

L1 – Lexical. Punctuation, whitespace, Unicode logical glyphs, and broken math shells are unified, and variant commands (`\land` → `\wedge`, etc.) are rewritten into a canonical form.

L2 – Syntactic. Every formula is parsed with a Lark grammar; well-formedness is checked at the level of brackets, operator scope, and predicate/-function structure. Failures route the item back for human repair rather than being silently dropped.

L3 – Semantic Alignment. Each symbol in a formula is checked for a consistent role, correct arity, and an explicit translation entry in the symbol gloss; this ties the formal language mechanically to the author's natural-language exposition.

L4 – Type and Parameter. Items are split into PL and FOL, deterministic variable renaming is applied to PL items, and minimal parameter regularization is applied to FOL items, producing the typed and parameter-normalized form that Z3 verification consumes.

B Adversarial Hardening Workflow

The hardening workflow (Section 2.2) is a five-step closed loop: Decider → Proposal → Review → Answering → Verification → finalize. The Proposal step is implemented by separate background and question proposers, and the Answering step is implemented by an answerer ensemble plus adjudication. Reviewer and Verifier act as gates: proposals that fail review or verification loop back to the proposers for revision, while answer-only failures trigger answer repair. Default role models and retry budgets are listed in Table 4. Only items that clear both gates are included in LLMEval-Logic-Hard.

B.1 Agent Roles

The five functional steps below summarize the workflow responsibilities; Table 4 and Figures 11–20 give the concrete model assignments and prompt-specialized implementations.

Agent role	Default model(s)
Decider	GPT-5.4 Pro
Background Proposer	Claude Opus 4.6
Question Proposer	Gemini 3.1 Pro
Reviewer	Gemini 3.1 Pro
Answerer ensemble	GPT-5.4 Pro, Claude Opus 4.6, Gemini 3.1 Pro
Adjudicator	GPT-5.4 Pro
Verifier	GPT-5.4 Pro
Answer repair	GPT-5.4 Pro
Per-item retry loop	**Max retries**
Review → Proposer	7
Verify → Proposer	5
Verify answer repairs	6

Table 4: Hardening-workflow configuration. *Top*: default model assignment for each agent role; the Answerer ensemble runs the three models in parallel and routes disagreements to the Adjudicator. *Bottom*: per-item retry budgets for the two design-rejection loops (Review / Verify → Proposer) and for the answer-repair loop triggered by the Verifier.

- **Decider** diagnoses the shallow solution path and emits hardening blueprints across all six strategies, grouped into background and question plans.
- **Proposal** rewrites the item in two prompt-specialized passes. The Background Proposer embeds the blueprint's mechanism network and dependency table, and the Question Proposer then writes sub-questions against the locked background using the fixed Hard question-type schema.
- **Review** audits candidate items before answering along three dimensions: background mechanisms, question responsibilities, and strategy coverage. It reports proposer-target issues using a fixed issue catalog.
- **Answering** uses a three-model answerer ensemble, with an adjudicator that arbitrates disagreements. Each answerer outputs one answer and one reasoning trace per sub-question under a unified answering protocol.
- **Verification** performs the final post-answer gate across answer correctness, reasoning validity, shallow-path leakage, and reviewer misses. It can trigger answer repair or send the item back to the proposers when answering evidence exposes residual design flaws.

Figures 11–20 show English translations of the system and user prompt pairs used by the hardening workflow.

B.2 Six Hardening Strategies

The hardening workflow applies six structural strategies, each required to demonstrably alter the closed candidate space, the required operation, or the reasoning path rather than rewording the surface text:

1. **add_branching** — Introduce multiple consequential branch axes, such as qualification status, time windows, rule-trigger conditions, evidence validity, presence, or priority order. Branches must inhabit the same state space, cross-constrain each other, and force recomputation, comparison, or incompatibility filtering across branches.
2. **add_distractor_premise** — Add distractor evidence that shares keywords, entities, time points, or resources with decisive evidence. Valid distractors must change exclusion paths, and at least one sub-question must explicitly depend on rejecting the distractor rather than treating it as background decoration.
3. **change_question_to_set_output** — Replace shortcut-prone single-point judgments with complete-search outputs or checks: full enumeration, counting, projection, uniqueness detection, alternative-solution search, or intersection/difference over feasible sets. The candidate space must be closed, and each sub-question should carry one primary output target rather than bundling enumeration, counting, comparison, and explanation into one query.
4. **add_uncertainty_or_multi_answer** — Introduce mechanism-grounded uncertainty through evidence attribution, missing observations, record anomalies, or display/state synchronization failures. The uncertainty must preserve multiple computable baseline worlds rather than collapsing into a single default branch, and should support both a direct uncertainty sub-question and a downstream propagation sub-question when possible.
5. **add_counterfactual_variant** — Add counterfactual sub-questions, rather than rewriting the baseline background, that flip one key fact or rule while holding all other conditions constant.

Metric	Mean	Max	$P(\geq 1)$
Proposer retries (total)	4.0	12	89.3%
Review→Proposer loops	2.9	8	80.8%
Verify→Proposer loops	1.1	6	58.0%
Verify answer repairs	1.2	6	58.9%

Table 5: Hardening gate convergence over $n = 224$ completed items. Proposer retries sum Review→Proposer (pre-answer design rejection) and Verify→Proposer (post-answer design rejection, distinct from answer repair which fixes the answer without changing the item). $P(\geq 1)$ is the fraction of items that trigger the metric at least once.

The hardening prompt asks for at least two distinct flips with distinct recomputation responsibilities, preferably over solution space, uniqueness, counts, or projections rather than a surface "does the conclusion change" check.

6. **alias_and_coreference_variation** — Assign multiple uniquely resolvable aliases to key entities, rule triggers, or resources. Alias resolution must impose real entity-tracking cost on at least one sub-question, rather than serving as surface-level synonym substitution or parenthetical one-to-one alignment.

B.3 Quality Control Statistics

Table 5 reports gate-level convergence statistics computed from the full per-item trajectory logs of 254 items that entered the hardening workflow, of which 224 completed (88.2%). The remaining 30 items exhausted the workflow retry budget before a passing revision was recovered and are excluded from the gate-level counters below. The $n = 224$ count is a workflow-completion statistic; final manual curation removed additional ambiguous or redundant items before release.

89.3% of items undergo at least one proposer retry (mean 4.0), and 80.8% trigger the Review→Proposer loop, confirming that the Reviewer gate enforces a non-trivial quality bar rather than rubber-stamping proposals. 58.0% of items are sent back to proposers by the Verifier *after* having passed the Reviewer, demonstrating that the two gates are complementary: pre-answer review alone cannot detect all design flaws that only surface once model answering traces expose shallow solution paths or unused background mechanisms. 58.9% of items require at least one answer repair, indicating that correct answer generation itself is iterative even when the item design is sound.

C Evaluation Detail

C.1 Answer Evaluation

Models generate structured JSON answers using a standardized prompt (Fig. 21). We then evaluate answers via LLM-as-judge: the judge receives the original problem text, the reference answer, and the model's answer, and decides semantic equivalence (Fig. 22). Two judge prompt variants handle multi-subquestion items and flat single-answer items respectively. The judge is instructed to treat the reference answer as the scoring target, accept harmless formatting or labeling differences, and reject only semantically contradictory or incomplete answers.

C.2 Translation Evaluation

All translation evaluation begins with a shared formalization step: the model translates a natural-language problem into solver-compatible Formal Language (FL) JSON. We use two formalization variants (Fig. 23): Free-FL, where the model declares all symbols, types, and translations from scratch; and Fixed-FL, where gold symbol declarations are provided as read-only context and the model writes only the premise set and solver queries. The resulting FL is then evaluated under one of two paradigms described below.

C.2.1 Z3 Mode

In Z3 mode, the candidate FL is fed to the Z3 solver, which executes each query under the declared premises and produces a structured model_answer (with supporting answer_tokens and answer_payload). An LLM then compares this solver-derived model_answer against the gold reference_answer for semantic equivalence (Fig. 24). The judge is instructed to accept formatting variations, stronger-positive answers for possibility questions, complement-based queries, and other structurally recoverable transformations when the reference answer can be obtained from the structured solver output.

C.2.2 Rubric Mode — Free Formalization

In rubric free-FL mode, the candidate FL is scored directly by an LLM against a per-problem rubric checklist with three item groups: logical_relation (LR), stated_constraint (SC), and query_alignment (QA). Each checklist item specifies a semantic requirement that the candidate formalization must satisfy. The LLM checks whether each item's requirement is met by the candidate's premises and queries

(Fig. 25). For QA items, the scorer uses LLM judgment when the optional `z3_check` does not establish query equivalence. LR/SC items are judged entirely by the LLM. The scoring prompt provides detailed rules for acceptable variants, equivalent rewritings, and boundaries between reasonable alternative formalizations and genuine errors. This mode evaluates formalization quality independently of answer correctness.

C.2.3 Rubric Mode — Fixed Formalization

In rubric fixed-FL mode, the same checklist structure is used, but with two additional mechanisms enabled by the shared symbol space. First, a Z3 premise-equivalence (PE) gate checks whether the candidate's premise set is logically equivalent to the gold premises; if equivalent, all LR and SC checklist items auto-pass. Second, when the PE gate reports non-equivalence but the candidate's Z3-solved answers still match the gold FL's solved answers for every gold query, an LLM soft-review re-evaluates each premise-level difference individually (Fig. 26). The LLM judges whether each extra or missing premise is a defensible formalization choice (softpass) or a genuine error (fail), using the original NL problem as authoritative and the checklist as secondary guidance. If all differences softpass, previously failed LR/SC items are promoted to pass. This hybrid approach uses Z3 to auto-resolve clear cases and reserves LLM review for ambiguous premise differences, catching genuine formalization gaps while avoiding false positives from reasonable alternative formalizations.

D LLM-as-Judge Validation

The leaderboard's Item / Sub-Q numbers (Table 2) and the formalization-side Z3 / Rubric / Both columns (Table 3) are both produced by LLM-as-Judge calls with `gpt-5.1-chat` as the backbone. To check that those verdicts reflect a stable evaluator rather than a single-judge artifact, we re-run two random samples with two additional frontier judges (Claude Opus 4.6 and Gemini 3.1 Pro) under the same answer- and rubric-judging prompts used in the main evaluation protocol, and report inter-judge agreement.

Setup. For the answer side, we draw 103 random sub-questions across all Hard run-1 model variants (21 items, sampling without replacement at the item level until the cumulative sub-Q count first reaches 100), and re-judge each item bundle with the three judges using the same bundled answer-judging prompt as in the main protocol (all sub-questions of an item judged in one JSON call; prompt shown in Fig. 22). For the formalization side, we draw 105 rubric atoms from the four publicly released Hy3 base score files (free / fixed $\times$ thinking / no-thinking) restricted to atoms that the rubric protocol delegates to the LLM, i.e., that are not auto-passed by the Z3 prefilter, since those are the only atoms whose verdict depends on the judge model. For each parent item, the three judges receive the same rubric-scoring prompt (Fig. 25) and the same payload (original problem, candidate FL, full rubric) and return a per-atom score. We use `temperature=0` for all three judges and disable Claude / Gemini's reasoning channel so the judge response is a deterministic JSON verdict.

Headline numbers. Table 6 reports the three pairwise Cohen's κ values and the unanimous-agreement rate on each side. All six pairwise κ's lie in $[0.873, 0.922]$ and therefore in the *almost perfect* band ($\kappa \geq 0.81$), comfortably above the $\kappa \geq 0.7$ threshold typically asked of judge-replacement studies in MT-Bench-style work; observed pairwise agreement is $\geq 94\%$ throughout, and all three judges return identical verdicts on 93.2% of sub-questions and 93.3% of atoms.

Limitations. The two cross-judges (Claude Opus 4.6 and Gemini 3.1 Pro) are themselves on the leaderboard, so this inter-judge agreement (IAA) does not rule out that all three judges share family-correlated blind spots; the 93% unanimous-agreement rate bounds the size of any such joint blind spot for the items we sampled but cannot eliminate it in principle. The rubric-side IAA further covers only the LLM-decided atoms, so it validates the judge model rather than the full hybrid rubric pipeline whose other atoms are settled deterministically by the Z3 prefilter.

E Why Z3 and Rubric Disagree

Two effects from Findings 2 and 3 are visible in Table 3: Free Z3 accuracy systematically exceeds Free Rubric accuracy, and even the best Fixed-symbols Rubric tops out at 72.36% instead of the 100% reached by the gold reference. Both follow from how each signal is wired rather than from a single failure mode in the candidate formalizations, and the two signals end up complementary in opposite directions.

Judge pair	Answer side (n=103 sub-Q)		Rubric side (n=105 atoms)	
	Obs. agree. (%)	Cohen's κ	Obs. agree. (%)	Cohen's κ
`gpt-5.1-chat` vs. Claude Opus 4.6	96.1	0.922	96.2	0.914
`gpt-5.1-chat` vs. Gemini 3.1 Pro	95.1	0.903	94.3	0.873
Claude Opus 4.6 vs. Gemini 3.1 Pro	95.1	0.903	96.2	0.914
all three judges unanimous	93.2		93.3	

Table 6: Inter-judge agreement on 103 randomly sampled Hard sub-questions (Answer side) and 105 randomly sampled rubric atoms restricted to those delegated to the LLM rather than auto-passed via Z3 (Rubric side). All three judges run at `temperature=0` on identical answer- or rubric-judging prompts; Claude / Gemini's reasoning channel is disabled to make the verdict deterministic. All six pairwise κ lie in the *almost perfect* band, $\kappa \geq 0.81$.

Z3 is binary and forgiving. Most LLMEval-Logic-Base items have a low-cardinality answer space: `possible` and `necessary` reduce to a binary decision, and `enumerate_models` typically resolves to a small finite set. Solver execution against the reference therefore admits a non-trivial chance of being right by accident whenever the candidate FL is wrong on the way to a right answer. We see three recurring patterns: (i) a candidate that over-constrains the premises still keeps the reference answer as one of its accidental solutions and is judged Z3-correct on the queried decision; (ii) a candidate that flips a logical relation (e.g., $\rightarrow$ vs $\leftrightarrow$) is masked by the polarity of the query, because both formalizations entail the same yes/no on that particular item; (iii) a candidate that omits a stated constraint is invisible to Z3 whenever the omitted constraint is not a binding constraint for the queried answer. In all three cases the rubric, which audits relations, stated constraints, and query alignment as separate atoms, registers a structural mismatch even when Z3 reports success. This matches the largest Free Z3−Rubric gaps in Table 3: GPT-5.4 Pro ($66.26 - 42.28 = 23.98$, the single largest gap), Qwen 3.5 Plus ($61.79 - 46.75 = 15.04$), and Claude Opus 4.6 ($64.63 - 50.41 = 14.22$). The candidate FLs diverge from the reference at the constraint or relation layer, but the divergence does not propagate to the solver's decision under the closed-world model space of the audited item.

Rubric is comprehensive but positive-only. Each rubric is a finite list of expected atoms over logical relations, stated constraints, and query alignment, with 5.69 atoms per item on average. Rubric pass rate therefore tells us whether the candidate FL contains the required semantics, not whether it contains *only* the required semantics. A candidate that adds an unsanctioned constraint, weakens a stated relation in a way that still satisfies the matched atom, or introduces a free variable not anticipated by the rubric will pass on every listed atom while encoding a substantively different formalization. The 72.36% Fixed-symbols Rubric ceiling reached by the strongest model is therefore an upper bound on rubric coverage rather than on faithful formalization: the residual $\sim$28 points combine genuinely missed atoms with errors that lie outside the rubric's positive coverage, and the latter component is invisible to both Z3 and Rubric individually.

Reading the combined metric. The two signals are complementary rather than redundant: Z3 acts as a hard filter for items where the answer space happens to expose the divergence, while Rubric audits what the candidate FL preserves. Their conjunction (the Both column in Table 3) is the closest single-shot estimator of faithful formalization that remains practical for free-form model outputs, but it still inherits both biases. The main obstacle is not merely the absence of a per-item equivalence check: in the free setting, models may introduce different symbols, predicates, type granularities, intermediate objects, and query decompositions, so candidate FLs often do not inhabit a reference-compatible formal space that Z3 can compare directly. Fixed-symbol evaluation partially removes this mismatch, but free-form formalization remains harder to align automatically. We therefore read the current 72.36% ceiling as a limitation of the evaluation instrument rather than evidence that the strongest models have saturated the formalization task; when reporting NL→FL faithfulness, the Both column should be tracked rather than Z3 alone.

Model	Base	CF	Δ (pp)
Gemini 3.1 Pro	77.70	71.56	-6.14
Claude Opus 4.6	78.23	72.22	-6.01
GPT-5.4 Pro	70.75	62.17	-8.58
Qwen 3.5 Plus	70.94	67.33	-3.61
Kimi K2.5	70.89	62.30	-8.59
Hy3 preview	66.18	60.19	-6.00
Seed 2.0 Pro	66.28	55.29	-10.99

Table 7: Hard Sub-Q Accuracy (%) on the baseline ($n = 686$) and counterfactual ($n = 252$) partitions of LLMEval-Logic-Hard for the seven thinking variants, averaged over 3 runs. Δ = CF − Base; every variant drops, isolating the counterfactual-recomputation cost from topic or wording effects.

F A Closer Look at Hard-Subset Difficulty

This section explains why the hardening strategies in Section 2.2 make the subset difficult, organising the analysis around three difficulty mechanisms. The key distinction from Error Analysis (Appendix G) is level of description: here we describe how the operators reshape the task; the following section shows how models fail on concrete cases.

F.1 Counterfactual recomputation

Counterfactuals force global re-derivation. The counterfactual variant operator replaces one fact or rule while holding the rest of the item fixed. This is deliberately different from surface paraphrase: a correct solver must reopen every branch whose status depended on the edited evidence and recompute the downstream model set. We partition the 938 Hard sub-questions into baseline ($n = 686$) and counterfactual ($n = 252$) by tagging each sub-question prefix for explicit counterfactual markers ("under the alpha / beta variant", "suppose X is changed to Y", etc.; partition script released alongside the dataset). Every thinking variant loses accuracy on counterfactual sub-questions (Table 7); the per-variant drop ranges from 3.6 pp for Qwen 3.5 Plus to 11.0 pp for Seed 2.0 Pro. Because the baseline and counterfactual questions sit inside the same items and share discourse style, the drop isolates the recomputation burden rather than topic or wording effects. The case studies in Appendix G show the corresponding failure pattern: models often patch the edited fact locally while retaining conclusions that were valid only in the original world.

F.2 Candidate-space maintenance and interference control

The non-counterfactual strategies enlarge and obscure what must be tracked. Branching creates multiple consequential axes that must be combined rather than solved independently; valid distractors introduce evidence that overlaps lexically with decisive evidence but must be rejected by rule-level reasoning; set-valued outputs turn a single decision into enumeration, counting, projection, uniqueness, or alternative-solution search over a closed candidate set; uncertainty requires maintaining multiple admissible worlds and propagating evidence tiers; and alias/coreference variation adds entity-tracking cost without changing the underlying logic. Together, these operators force the model to maintain a complete candidate space and to filter interference, so difficulty rises even when each individual logical relation remains locally simple.

F.3 Multi-question state maintenance

Hard items are not independent questions over independent worlds. The first sub-question typically anchors the closed model space, and later sub-questions query projections, alternatives, counterfactual updates, or evidence tiers over that same space, so the model needs both local logical encoding and cross-question state maintenance. Strict item scoring then magnifies any slip: Hard items average 4.94 sub-questions and Item Accuracy requires every one of them to be correct, so a single under-enumerated branch, accepted distractor, or missed counterfactual update collapses the item-level score even when most local relations are encoded correctly. Reporting Base, Hard Sub-Q, and Hard Item scores jointly therefore separates three different competences: Base captures single-shot encoding, Sub-Q captures local partial credit, and Item Accuracy captures sustained all-correct control over the multi-question model space.

Practical guidance. For model diagnosis, these three mechanisms support targeted ablations. A baseline-vs-counterfactual split tests global recomputation; set-valued and uncertainty / distractor questions test exhaustive search and interference control; and alias/coreference questions test entity tracking. Reporting these slices alongside the aggregate Hard score makes it possible to distinguish a model that lacks local logical encoding from one that encodes local relations but fails to maintain the candidate space across the item.

G Error Analysis

We analyze common failure patterns of the 14 evaluated models on LLMEval-Logic-Base and LLMEval-Logic-Hard. Because every item provides all required background facts, the dominant errors are not knowledge gaps; they are failures to preserve formal semantics, maintain complete model sets, and recompute dependent conclusions under counterfactual changes. We organize the taxonomy into two Base error types and four Hard error types, and retain only case studies whose reference answers are stable under re-audit.

G.1 Base Errors

Modal quantifier tracking failure. The most frequent Base failures come from confusing existential and universal queries over the feasible model set: a statement that holds in *some* model ($\Diamond$) is promoted to one that holds in *all* models ($\Box$), or a forced conclusion is downgraded to uncertainty. This also appears as a rule-semantic translation error: models answer a validity question even when the task asks whether a conclusion is merely satisfiable. Figure 7 gives a representative case. The symmetry is visible at scale, with high-frequency errors in both $\Diamond \rightarrow \Box$ and $\Box \rightarrow \Diamond$ directions rather than a one-sided bias.

Incomplete model enumeration. The second Base failure is incomplete traversal of the constraint space: models replace the complete set of feasible assignments with the first coherent explanation, prune legal branches too early, or merge independent uncertainty dimensions. This explains why `enumerate_models` and `count_models` are especially fragile: once a model set is under-enumerated, the final list and count are both necessarily wrong.

G.2 Hard Errors

Counterfactual recomputation failure. The largest Hard failure source is local counterfactual editing without global recomputation. Models replace one fact, keep downstream conclusions that were derived from the old fact, and then over-constrain the new world. Figure 8 shows a case where the edited meeting fact must be recomposed with surviving branches. Correct solving requires reopening every branch whose status depended on the edited evidence, not simply patching the surface description.

Complete model enumeration failure. Hard enumeration failures involve multiple interacting axes of uncertainty, each with its own evidence tier or validity condition. Models often enumerate one axis at a time but fail to preserve legal cross-products across axes; Figure 9 shows this on a state-tuple enumeration task. Because the first Hard sub-question usually defines the baseline model set, this error cascades into later projection, comparison, and counterfactual sub-questions.

Set-family and projection operation errors. Many Hard tasks operate over nested structures: elements, sets of elements, and families of sets. Models frequently compute at the wrong level, e.g., classifying individual elements when the question asks for set-family members, or performing a formal set difference but failing to map the resulting symbols back to the semantic labels required by the question. Figure 10 shows this projection-level failure.

Evidence-provenance and validity-tier maintenance failure. Hard items add evidence hierarchies absent from Base: formal records, marginal notes, ambiguous annotations, destroyed confirmations, and source-dependent validity. Models often elevate weak evidence to decisive proof, retain evidence chains invalidated by a counterfactual, or compress structured candidates in a way that loses the binding between evidence source and conclusion.

H Human Annotation Details

Human involvement in LLMEval-Logic was limited to benchmark construction and audit. Contributors were students at our institution with at least one year of training in mathematical logic; they wrote self-contained, fictional Chinese reasoning problems, avoided personal or sensitive data, and provided initial reference formalizations and answers. Annotators were professionals with multiple years of mathematical-logic research experience; they audited clarity, reasoning validity, NL–FL faithfulness, and rubric atoms under Section 2.1. Both groups received partial subsidies and compensation for their work. Before contributing, they were informed that the resulting items, formalizations, rubrics, and audit traces would be used for research, and they consented to that use.

I Agent and Judge Prompts

This appendix collects the verbatim system and user prompts used in the hardening and evaluation protocols: the five hardening-agent prompts referenced in Appendix B, and the model-generation and LLM-as-Judge prompts referenced in Appendix D. Chinese prompts have been translated for readability while preserving the exact field structure and JSON schema; the original Chinese strings are shipped with the released code.

ID #73 Modal Quantifier Tracking Failure (possible)

Background. Let A denote "watch Top Esports in international matches," and let B denote "T1 reaches the final." The item gives two premises: $A \to B$ and $B = \text{true}$.
Question. Is it possible to conclude $\neg A$ from these premises?

Reference Answer & Correct Reasoning

Answer. Yes, $\neg A$ is possible.
Reasoning. The premises do not force A to be true. Both assignments $A = \text{true}, B = \text{true}$ and $A = \text{false}, B = \text{true}$ satisfy $A \to B$ and B. Therefore $\neg A$ is satisfiable under the premise set. The task asks a $\Diamond$ query over feasible models, not whether $\neg A$ is entailed in all feasible models.

Gemini 3.1 Pro

Answer. No.

Reason (translated). $A \to B$ and B true is affirming the consequent; formal logic cannot necessarily infer A, so it cannot infer $\neg A$.

Error. Answers an entailment query instead of the possibility query.

Claude Opus 4.6

Answer. Cannot derive $\neg A$.

Reason (translated). B being true is compatible with both A true and A false; therefore there is no logical basis to derive $\neg A$.

Error. Recognizes compatibility but still reports non-derivability as failure.

GPT-5.4 Pro

Answer. Cannot derive $\neg A$.

Reason (translated). Both $(A = \text{true}, B = \text{true})$ and $(A = \text{false}, B = \text{true})$ satisfy the premises, so $\neg A$ cannot be derived.

Error. Its reason explicitly exhibits a $\neg A$ model, but the final answer treats this as non-entailment.

Figure 7: **Case study of modal quantifier tracking failure (Base ID #73).** All 14 models answer the validity question rather than the satisfiability question, producing the wrong binary decision even though their local entailment statement is formally sound.

ID #359.4 Counterfactual Set-Family Recomposition Failure (hard/direct basis set)

Background. The speaker decides whether to attend tomorrow's class reunion. Rule 1: if Xiaomai's status is "definitely going," the speaker goes regardless of whether tomorrow has an important meeting. Rule 2: if Xiaomai's status is "definitely not going," tomorrow has no important meeting, and many old friends will attend, the speaker goes. Rule 3: if the venue is close to home and tomorrow has no important meeting, the speaker goes even without considering Xiaomai or old friends. Rule 4: if the reunion is crowded and Xiaomai is "definitely not going," the speaker definitely does not go. Rule 5: if Rules 1–4 cannot derive a conclusion, the speaker does not go. Rule 4 is an absolute veto when its antecedent holds; otherwise, any true antecedent among Rules 1–3 is sufficient for going. The "direct basis set" of a complete model is the set of positive rule numbers whose antecedents are true; if no positive antecedent is true, it is $\emptyset$ and Rule 5 derives not going. Xiaomai is not treated as an open fact of whether she actually attends, but as a closed three-valued status: definitely going, definitely not going, or unlocked. Rule 1 fires only on definitely going; Rules 2 and 4 fire only on definitely not going. Evidence $E1$: a cached signup screenshot includes "Mai-jie," but the system may still display cancelled signups. $E2$: roommate Zheng reports that "the sophomore monitor will definitely come tomorrow," and the sophomore monitor is Xiaomai, but Zheng may have confused it with the department event the day after tomorrow. $E3$: organizer Ning says she has just removed Mai-jie from the list, but also admits she was processing two lists and the removal may have been for the department dinner the day after tomorrow. Thus Xiaomai may remain in any of the three statuses. Crowding is determined only by the organizer's final venue message: after an earlier warning, the final message says that adding two large tables gives everyone a formal seat, so the reunion is not crowded at baseline. An important meeting exists iff the calendar contains a formal meeting explicitly marked important or indispensable: the 9 a.m. weekly meeting is not important, but the 3 p.m. client call is marked "important, cannot miss," so baseline has an important meeting. The venue is Old Place Hotpot, an eight-minute walk from home, so it is close. "Many old friends" means at least three confirmed attendees are old friends. The closed contact list fixes Chen Siyuan and Zhou Yang as old friends, Sun Ting as not, while "Jiaqi" is either Li Jiaqi (not an old friend) or Li Jiaqi with a different character (an old friend), and "A-Lei" is either Zhao Lei (not an old friend) or Wu Lei (an old friend).
Question. In world α, the afternoon client call is cancelled and there is no other important meeting. List all possible direct basis sets.

Reference Answer & Correct Reasoning

Answer. $\{\{\text{Rule } 1, \text{Rule } 3\}, \{\text{Rule } 3\}, \{\text{Rule } 2, \text{Rule } 3\}\}$.
Reasoning. Cancelling the important meeting makes Rule 3 true in every branch because the venue remains close. If Xiaomai is definitely going, Rule 1 also fires. If Xiaomai is definitely not going and at least one alias branch adds a third old friend, Rule 2 also fires; otherwise only Rule 3 fires. Rule 4 remains blocked because the baseline final venue message says the reunion is not crowded.

Gemini 3.1 Pro

Answer. $\{\{\text{Rule } 3\}\}$.

Reason (translated). The meeting cancellation activates Rule 3; Xiaomai remains unlocked, so Rules 1 and 2 do not fire.

Error. Fails to propagate the surviving Xiaomai and old-friend branches after editing the meeting axis.

Claude Opus 4.6 (no-think)

Answer. $\{\{\text{Rule } 3\}\}$, as the unique value.

Reason (translated). With no important meeting, Rule 3 necessarily triggers; Xiaomai is still unlocked, so all Xiaomai-dependent rules fail.

Error. Recomputes only the edited meeting fact and keeps the previous over-collapsed Xiaomai state.

Qwen 3.5 Plus

Answer. $\{\{\text{Rule } 3\}\}$.

Reason (translated). Alpha has no important meeting, so Rule 3 fires; Xiaomai is unlocked, so Rules 1 and 2 remain false.

Error. Drops both $\{\text{Rule } 1, \text{Rule } 3\}$ and $\{\text{Rule } 2, \text{Rule } 3\}$ by not reopening the unresolved branches.

Figure 8: **Case study of counterfactual set-family recomposition failure (Hard ID #359.4).** Correct solving requires editing the important-meeting fact while preserving all independent branches over Xiaomai's status and old-friend aliases.

ID #252.1 Complete Enumeration Failure (hard/state tuples)

Background. A core laboratory is controlled by one central controller. The item tracks exactly five state dimensions: whether an intrusion alarm has triggered, whether fingerprint biometric verification has passed, whether the system is locked, whether password verification has passed, and whether the system has started. Specification $S1$: if the system has started, both fingerprint and password verification must already have passed; when the system is not locked, both verifications may have passed even if the operator has not yet pressed start. Specification $S2$: the locked state can be caused only by an intrusion alarm. Specification $S3$: once an intrusion alarm triggers, the system immediately enters locked state and simultaneously revokes fingerprint pass, password pass, and started status; without an administrator unlock record, those three states cannot become passed/started again. No administrator unlock is recorded. There are two displays. The host panel shows the host's current state in real time. The lobby mirror display synchronizes a host snapshot every five minutes and keeps the previous snapshot between synchronizations. The two displays use the same visual format, but the guard only moves in the lobby, so every display he sees is the mirror display. A short lobby beep has exactly two possible sources: a scheduled mirror synchronization, or an intrusion alarm just triggered through the lobby linkage. Records are: A, the 18:00–22:00 previous shift ended with the system normal and no alarm; B, a 22:15 smoke-sensor false alarm in the outer visitor area was unrelated to the core lab; C, at 22:40 the guard saw the lobby mirror showing green "started" with both verifications passed, but did not know which synchronization snapshot it reflected; D, the guard did not enter the lab and did not know the host-panel state at 22:40; E, at 22:45 the guard heard one short lobby beep with ambiguous source; F, at 22:50 the guard saw the lobby mirror immediately after its 22:50 scheduled synchronization, and it still showed green "started" with both verifications passed. When enumerating the 22:46 model, the tuple is $(alarm, biometric, locked, password, started, 22{:}45\ beep\ source)$; if the beep source is alarm-trigger sound, then by 22:46 the system must already be locked and the three pass/start states must be revoked.
Question. Using only records A–E, enumerate all feasible 22:46 six-tuples.

Reference Answer & Correct Reasoning

Answer. Six models: $(N, N, N, N, N, sync)$, $(N, Y, N, N, N, sync)$, $(N, N, N, Y, N, sync)$, $(N, Y, N, Y, N, sync)$, $(N, Y, N, Y, Y, sync)$, and $(Y, N, Y, N, N, alarm)$.
Reasoning. If the 22:45 beep is the alarm trigger, $S2/S3$ force the unique locked tuple. If it is the scheduled synchronization sound, records A–E do not constrain the 22:46 host state beyond $S1$ and the absence of alarm evidence, leaving exactly five non-locked combinations: no verification, biometric only, password only, both verifications without start, and both verifications with start.

Gemini 3.1 Pro

Answer. $(N, Y, N, Y, Y, sync)$, $(Y, N, Y, N, N, sync)$, $(Y, N, Y, N, N, alarm)$.

Reason (translated). The model keeps the green mirror state as one branch and treats the locked state as possible under either beep source.

Error. Omits four legal non-alarm states and adds an impossible sync-labeled lock tuple.

Claude Opus 4.6

Answer. Three models: green-started sync, locked sync, and locked alarm.

Reason (translated). It assumes the system stays started unless an alarm occurs, and allows a non-22:45 alarm while the 22:45 beep is synchronization.

Error. Treats the stale mirror image as current host evidence and invents an unobserved alarm path.

GPT-5.4 Pro

Answer. The six reference models plus $(Y, N, Y, N, N, sync)$.

Reason (translated). It correctly enumerates the five non-alarm $S1$ branches but keeps a locked branch with synchronization as the beep source.

Error. Adds a closed-world alarm state without the only admissible alarm-trigger evidence.

Figure 9: **Case study of complete-enumeration failure (Hard ID #252.1).** The model must enumerate the five non-alarm states allowed by $S1$ and the single alarm-trigger state, instead of treating the stale mirror snapshot as the current host state. Y/N abbreviate yes/no; *sync* and *alarm* abbreviate the two possible 22:45 beep sources.

ID #359.3 Projection-Level Set Family Failure (hard/direct basis set)

Background. The speaker decides whether to attend tomorrow's class reunion. Rule 1: if Xiaomai's status is "definitely going," the speaker goes regardless of whether tomorrow has an important meeting. Rule 2: if Xiaomai's status is "definitely not going," tomorrow has no important meeting, and many old friends will attend, the speaker goes. Rule 3: if the venue is close to home and tomorrow has no important meeting, the speaker goes even without considering Xiaomai or old friends. Rule 4: if the reunion is crowded and Xiaomai is "definitely not going," the speaker definitely does not go. Rule 5: if Rules 1–4 cannot derive a conclusion, the speaker does not go. Rule 4 is an absolute veto when its antecedent holds; otherwise, any true antecedent among Rules 1–3 is sufficient for going. The "direct basis set" of a complete model is the set of positive rule numbers whose antecedents are true; if no positive antecedent is true, it is $\emptyset$ and Rule 5 derives not going. Xiaomai is not treated as an open fact of whether she actually attends, but as a closed three-valued status: definitely going, definitely not going, or unlocked. Rule 1 fires only on definitely going; Rules 2 and 4 fire only on definitely not going. Evidence $E1$: a cached signup screenshot includes "Mai-jie," but the system may still display cancelled signups. $E2$: roommate Zheng reports that "the sophomore monitor will definitely come tomorrow," and the sophomore monitor is Xiaomai, but Zheng may have confused it with the department event the day after tomorrow. $E3$: organizer Ning says she has just removed Mai-jie from the list, but also admits she was processing two lists and the removal may have been for the department dinner the day after tomorrow. Thus Xiaomai may remain in any of the three statuses. Crowding is determined only by the organizer's final venue message: after an earlier warning, the final message says that adding two large tables gives everyone a formal seat, so the reunion is not crowded at baseline. An important meeting exists iff the calendar contains a formal meeting explicitly marked important or indispensable: the 9 a.m. weekly meeting is not important, but the 3 p.m. client call is marked "important, cannot miss," so baseline has an important meeting. The venue is Old Place Hotpot, an eight-minute walk from home, so it is close. "Many old friends" means at least three confirmed attendees are old friends. The closed contact list fixes Chen Siyuan and Zhou Yang as old friends, Sun Ting as not, while "Jiaqi" is either Li Jiaqi (not an old friend) or Li Jiaqi with a different character (an old friend), and "A-Lei" is either Zhao Lei (not an old friend) or Wu Lei (an old friend).
Question. Combining the previous two projections with the remaining baseline facts, list all possible values of the baseline direct basis set.

Reference Answer & Correct Reasoning

Answer. $\{\{\text{Rule } 1\}, \emptyset\}$.
Reasoning. At baseline the important meeting blocks Rules 2 and 3, and the final venue message blocks the veto Rule 4. If Xiaomai is definitely going, Rule 1 is the sole positive direct basis; if Xiaomai is definitely not going or unlocked, no positive rule antecedent is true and the direct basis is $\emptyset$.

Gemini 3.1 Pro

Answer. $\{\emptyset\}$.

Reason (translated). It says Xiaomai remains unlocked and the important meeting blocks all positive rules.

Error. Collapses the Xiaomai tri-state branch and misses the model where Rule 1 fires.

Claude Opus 4.6 (no-think)

Answer. $\{\emptyset\}$, as the unique value.

Reason (translated). It fixes Xiaomai to unlocked, so Rules 1, 2, and 4 cannot trigger; Rule 3 is blocked by the important meeting.

Error. Turns one compatible Xiaomai status into the only status and discards the set-family member $\{\text{Rule } 1\}$.

Qwen 3.5 Plus

Answer. $\{\emptyset\}$.

Reason (translated). Xiaomai is unlocked, so Rules 1, 2, and 4 fail; the important meeting blocks Rule 3.

Error. Treats "unlocked" as the only surviving Xiaomai value instead of one member of the closed three-valued status set.

Figure 10: **Case study of projection-level set-family failure (Hard ID #359.3).** Correct solving requires projecting each complete baseline model to its direct-basis set and deduplicating the resulting family of sets.

System Prompt — Decider (initial)

You are the ‘initial_decider‘ for LLMEval-Logic reason-mode hardening.
Your task is not to make lightweight "difficulty-increase suggestions," but to directly design an initial hardening blueprint that meets benchmark hard-case standards for the given item.
You must: - Design hardening only at the natural-language level; do not modify the formalization. - Use ‘background_strategy_plans[]‘ and ‘question_strategy_plans[]‘ as the primary outputs. - Ensure that all six strategies appear in the union of the two plan blocks; the same strategy may appear in both blocks. - Also output a ‘strategy_blueprints‘ compatibility mirror, but it is only a compatible copy of the grouped plans—not a space for free-form additions. - Output only a JSON object strictly matching the schema. Do not output prose, markdown, explanatory preambles, or code blocks.

User Prompt — Decider (initial)

Strategy Catalog Overview: {{strategy_catalog}}
Your task is to directly design an initial hardening blueprint at the benchmark hard-case level for this item, not merely "make it a bit harder than the original."
Working rules: - Treat grouped plans as the primary output: - ‘background_strategy_plans[]‘ - ‘question_strategy_plans[]‘ - The union of both plan blocks must cover all six strategies; the same strategy may appear in both. - Every question set should reserve at least one high-quality mechanism-grounded uncertainty anchor and try to have it affect at least two qids: - one ‘direct uncertainty qid‘ - one ‘propagation qid‘ - Both qid types must continue to fall within the existing seven semantic question types; do not design uncertainty as an additional new question type. - ‘background_strategy_plans[]‘ focuses on: what key clues to write in the background, how these clues couple, and which mechanisms must enter the background to be genuinely hard. - ‘question_strategy_plans[]‘ focuses on: what hard-case responsibilities the question set should carry, what question forms to prefer, and what shallow implementations to avoid. - The question-set blueprint must anticipate that "each qid should correspond as much as possible to a single independent query"; do not default to squeezing multiple responsibilities into one sub-question. - Clearly distinguish: - single comparison tasks that can be kept as one qid - multi-task packaged questions that must be split into multiple qids - ‘strategy_blueprints‘ must align with grouped plans and be output as a compatibility mirror; do not add free-form content in ‘strategy_blueprints‘ that is absent from grouped plans. - You are not a proposer. Do not write full background or full question text; write only blueprints and responsibilities. - Do not design hard cases as "more questions" or "longer background"; the focus is mechanism intersection, full-enumeration cost, counterfactual recomputation, coreference burden, and distractor elimination cost. - Do not concentrate the hard-case burden onto a single over-weighted sub-question while letting other sub-questions play filler or coverage roles.
Your criteria for benchmark hard case: - The item must not be dominated by a single master rule or a single temporal threshold. - The question set must contain multiple non-isomorphic high-cost responsibilities, not reskinned versions of the same state space. - Richer-output must target a closed candidate space; open-class targets do not qualify as hard-case set-output. - Multi-solution or uncertainty must arise from the mechanisms themselves, not from rule gaps, undefined priorities, or specification holes. - Mechanism-grounded uncertainty should preferentially use the following archetypes: - evidence provenance uncertainty - missing observation does not equal false - record/display anomaly causing indeterminate state - Distractors, aliases, and branching must genuinely change the solution space or reasoning path, not serve as decoration. - Each sub-question should map as naturally as possible to a single independent query; if one responsibility requires more than two independent conclusions, split into multiple qids by default.
Below is a detailed exposition of the six strategies under hard-case criteria. You must implement these standards in the grouped plans.

Figure 11: System and user prompts for the Decider agent (initial, translated).

User Prompt — Decider (initial, continued)

‘add_branching‘ - Hard-case implementation in the background: - Form at least several key clues that genuinely change the solution space and make them cross-constrain each other. - Branch axes should preferentially come from qualification status, time windows, rule-trigger conditions, evidence validity, presence status, or priority order. - Branches must inhabit the same state space, not have each question independently invoke one rule each. - Hard-case implementation in questions: - Questions should force solvers to recompute, compare, or filter out incompatible cases across multiple branches. - At least one question type should involve cross-branch recomputation, before/after state comparison, or solution-space change judgment induced by branching. - These responsibilities should be distributed across multiple independent qids, not packed into one compound question. - False implementation / shallow forms: - Merely adding a rule list to the background while most rules do not affect the answer. - Each question independently invokes one axis, so the overall remains local judgment.
‘add_distractor_premise‘ - Hard-case implementation in the background: - Distractors must share keywords, entities, time points, or resources with decisive evidence, making them highly relevant on the surface. - Distractors should change exclusion paths, forcing models to identify information that "looks relevant but is insufficient to determine the answer." - Hard-case implementation in questions: - At least one question type must explicitly depend on the "exclude the distractor" step during answering; otherwise the distractor is mere background decoration. - Distractor exclusion should serve as the single primary responsibility or core solving step of a qid, not a supplementary remark. - False implementation / shallow forms: - Completely unrelated life-style descriptions. - Distractors that directly rewrite a main rule, becoming a new main rule. - Distractors that, although relevant, change no question’s solution space when deleted.
‘change_question_to_set_output‘ - Hard-case implementation in the background: - The background should, as much as possible, close the candidate space through type boundaries and rule boundaries, serving enumeration, counting, projection, uniqueness, and alternative-solution questions. - Hard-case implementation in questions: - Questions must require full enumeration, projection, counting, uniqueness, or alternative detection. - Richer-output must not merely lengthen the answer form; it must force solvers to traverse the full candidate space. - Prioritize designing questions that list all feasible cases, count quantities, compute intersection/difference, judge uniqueness, or find alternatives. - A single qid bears only one primary output target; do not simultaneously require "enumerate all + count + compare + explain" in one question. - False implementation / shallow forms: - Turning set questions or uniqueness questions back into plain yes/no. - Candidate space is not closed, yet the question asks to "list all." - Listing only one or two obvious objects without actually performing full search.
‘add_uncertainty_or_multi_answer‘ - Hard-case implementation in the background: - Multi-solution, non-uniqueness, or temporary indeterminacy must arise from the background mechanisms themselves. - Such uncertainty should co-operate with branching, set-output, counterfactual, and other mechanisms. - Each question set should reserve at least one uncertainty anchor, forcing models to retain multiple possible worlds rather than collapsing unknown directly into false/invalid/absent. - Prioritize the following high-efficiency archetypes: - evidence provenance uncertainty, e.g., messages, certificates, oral statements, or record sources unresolved - missing observation does not equal false, e.g., scan blind spots, unrecorded states, unobserved attributes - record/display anomaly causing indeterminate state, e.g., display panel fault, missing register page, state sync failure - Hard-case implementation in questions: - Questions should include at least 1 ‘direct uncertainty qid‘ that directly outputs multi-solution sets, non-uniqueness status, or possibilities still retained under the mechanism. - The same uncertainty anchor should also try to affect 1 ‘propagation qid‘, for example: - recompute the determinable feasible set under the same uncertainty - recompute objects that remain determinable/indeterminable after counterfactual - judge whether uncertainty changes uniqueness, candidate count, or projection result - ‘direct uncertainty qid‘ preferentially maps to: necessary, unique_solution, has_alternative, enumerate_models, projection - ‘propagation qid‘ preferentially maps to: enumerate_models, projection, count_models, unique_solution - Uncertainty questions must still be able to explain why multiple solutions exist, rather than treating "missing information" directly as the answer. - If a multi-solution judgment and another independent task co-occur, split into separate questions by default. - False implementation / shallow forms: - Using rule gaps, undefined priorities, or specification holes to manufacture "indeterminacy." - Adding a single weak tie-breaking rule that barely increases search cost. - Having only one isolated "indeterminate" question while the same mechanism has no propagation effect on any other qid.
‘add_counterfactual_variant‘ - Hard-case implementation in the background: - Do not write counterfactuals back into the background body; the background provides only the baseline mechanism. - Hard-case implementation in questions: - Counterfactual questions must independently recompute on the original baseline. - Prioritize flipping only one key fact or one key rule while holding all other conditions constant. - Demonstrate at least 2 different flips with distinct recomputation responsibilities. - Counterfactual questions should preferentially require recomputing solution space, uniqueness, count, or projection, rather than merely asking "did the conclusion change?" - If one counterfactual question requires more than two independent conclusions, split into multiple qids by default. - False implementation / shallow forms: - Packaging multiple flips into one vague large modification. - Using counterfactuals to wrap repeated judgments of the same master rule.
‘alias_and_coreference_variation‘ - Hard-case implementation in the background: - Key entities, rule triggers, or resources should have multiple uniquely resolvable names. - Mapping relationships should first be established in the background, then invoked in the questions. - Hard-case implementation in questions: - At least one question type must genuinely depend on alias or coreference resolution to be answered correctly, not obtainable at a glance after parenthetical alignment. - False implementation / shallow forms: - Merely inconsequential word substitutions. - Direct parenthetical character-by-character alignment imposing almost no entity-tracking cost. - Causing genuinely irresolvable ambiguity.
Output requirements: - ‘weakness_summary‘: summarize why the original item is insufficient to constitute a hard case. - ‘evidence[]‘: provide evidence supporting this judgment; do not write empty evaluations. - ‘background_strategy_plans[]‘: each entry contains ‘strategy‘, ‘strength‘, ‘background_guidance‘. - ‘question_strategy_plans[]‘: each entry contains ‘strategy‘, ‘strength‘, ‘question_guidance‘. - ‘strategy_blueprints[]‘: generate a compatibility mirror from grouped plans, each entry carrying both ‘background_guidance‘ and ‘question_guidance‘. - ‘expected_failure_traps[]‘: list the most likely shallow paths or misjudgment paths the model will take. - ‘target_models‘, ‘reason_mode_only‘, ‘nl_only‘ must be filled in according to item semantics; do not omit.
‘background_guidance‘ focuses on: ‘key_clues‘, ‘clue_couplings‘, ‘execution_notes‘. ‘question_guidance‘ focuses on: ‘responsibilities‘, ‘preferred_question_shapes‘, ‘forbidden_shallow_forms‘. ‘question_guidance.responsibilities‘ must be written as: the single primary responsibility a qid should bear; if multiple responsibilities are needed, explicitly state that they must be split into different qids. In ‘question_guidance.preferred_question_shapes‘, comparison questions may be retained provided the comparison itself is the sole primary task, not a packaged question that "answers two worlds separately and then outputs a third judgment."
Do not mechanically pursue quotas, but ‘add_counterfactual_variant‘ must demonstrate at least 2 different flips. The output must strictly match the ‘response_format‘ schema.
target models: {{target_models_json}}
config: {{config_json}}
item: {{item_json}}

Figure 11: System and user prompts for the Decider agent (initial, translated; continued).

System Prompt — Background Proposer (initial)

You are the ‘background_proposer_initial‘ for LLMEval-Logic hardening.
Your task is to first construct the background mechanisms needed for a benchmark hard case, then write them into a complete background with a lightweight dependency table.
You must: - Write only the background; do not write questions or answers. - Write the background as a mechanism network that genuinely affects the solution space, not as "bulk expansion." - Output complete ‘background‘, ‘background_dependency_table‘, and ‘notes‘, and output only structured JSON.

User Prompt — Background Proposer (initial)

Strategy Catalog Overview: {{strategy_catalog}}
Based on the original item and the ‘initial_decider‘ blueprint, generate the first version of the hardening background. Your goal is not to lengthen the background but to first build out the item’s hard-case mechanisms.
Working rules: - First construct around ‘background_strategy_plans[]‘, then ensure these background mechanisms can support the high-cost question types required by ‘question_strategy_plans[]‘. - The background must form a network of key clues that genuinely change answers, candidate sets, or the solution space. - Each key background clue should change at least one category of downstream sub-question’s answer, candidate space, or exclusion path. - Each question set should embed at least 1 mechanism-grounded uncertainty anchor in the background, and make it genuinely change the candidate set, rule triggering, or state classification within the closed state space. - Uncertainty anchors should preferentially use these archetypes: evidence provenance uncertainty, missing observation does not equal false, record/display anomaly causing indeterminate state. - The background should support "each qid corresponds as much as possible to a single independent query"; do not build difficulty on terminological ambiguity, role-boundary vagueness, or rule gaps. - Candidate spaces should be closed preferentially through type boundaries and rule boundaries, not by exhaustively listing entity rosters. - If an uncertainty anchor falls on qualification records, slot assignments, or quota allocation, simultaneously write the formal assignment rules; do not only write who "may be qualified." - If finite resource slots exist, specify whether the sole qualified candidate necessarily locks the slot, and when vacancy is permitted. - ‘add_distractor_premise‘ must change exclusion paths, not be merely ineffective noise sharing keywords. - ‘alias_and_coreference_variation‘ must raise reasoning localization cost for at least one category, not be mere parenthetical word substitution. - If a background rule would only create disputes like "does this identity count as X" or "does this access right count as an occupancy right" without increasing full-search or recomputation cost, it should not be retained.
Explicitly prohibited: - Manufacturing uncertainty through rule gaps, undefined priorities, or specification holes. - Writing only the necessary condition for occupying a resource slot without specifying whether qualification leads to formal slot locking. - Adding decorative aliases. - Adding "background noise" that affects no question when deleted. - Letting the background remain fully dominated by a single master rule. - Creating fake difficulty by deliberately blurring concepts like "qualification right / access right / occupancy right / certification right." - Writing a fake uncertainty anchor that appears to carry uncertainty wording but actually does not affect any downstream qid.
Output requirements: - ‘background‘: complete background body. - ‘background_dependency_table‘: retain only the most critical dependency entries. Each entry writes: ‘clue‘, ‘why_it_matters‘, ‘question_hooks‘. - At least one dependency must explicitly correspond to the uncertainty anchor and explain which qids or which of the seven question-type forms it affects. - ‘notes‘: record the hard-case design priorities of this background version; do not write filler. - Do not modify the formalization. - Output must strictly match the ‘response_format‘ schema.
item: {{item_json}}
initial decision: {{initial_decision_json}}
existing background proposal: {{existing_background_proposal_json}}
revise decision: {{revise_decision_json}}

Figure 12: System and user prompts for the Background Proposer agent (initial, translated).

System Prompt — Question Proposer (initial)

You are the ‘question_proposer_initial‘ for LLMEval-Logic hardening.
Your task is to produce a set of benchmark hard-case questions after the background is locked, not merely to outfit the background with questions that "look more complex."
You must: - Generate a set of self-contained, few-but-hard, non-isomorphic sub-questions. - Output minimal responsibility evidence for each question: ‘role‘, ‘background_dependencies‘, ‘strategy_tags‘. - Do not compress multiple high-cost responsibilities into a few shallow question slots.

User Prompt — Question Proposer (initial)

Strategy Catalog Overview: {{strategy_catalog}}
Based on the locked background and the ‘initial_decider‘ blueprint, output the first version of the complete question set. Your primary goal is to produce a benchmark hard-case question set; strategy coverage is the secondary goal.
Working rules: - The question-set style is fixed as "few but hard"; do not pursue question count. - Each question must be self-contained; it must not reference a previous question or share local conditions without restating them. - Each qid may have only one primary output target and should correspond as much as possible to a single independent query. - Every question set should contain at least 1 direct uncertainty qid, and should try to also contain 1 propagation qid affected by the same uncertainty anchor. - Both types of qid must continue to fall within the existing seven semantic question types; do not write uncertainty as a vague open question. - Maintain extremely low tolerance for isomorphic questions: as long as two questions mainly reuse the same master rule or reskin the same state space, they are disqualified. - Strong strategies may be concentrated in a small number of questions, but must not be only shallowly displayed; if the overall still fails to genuinely unfold, the set is disqualified. - Richer-output questions must target a closed candidate space; open-class targets such as "any other student" or "any random person" do not qualify as hard-case set-output. - Set questions, count questions, projection questions, uniqueness questions, alternative-solution questions, and counterfactual questions must all genuinely require full enumeration, recomputation, or before/after state comparison. - Mechanism-grounded uncertainty should preferentially land in the following direct qid forms: - which objects cannot be determined - whether unique / whether non-unique - whether there are other feasible solutions - list all possible solutions - give the set under ‘possible‘ / ‘necessary‘ semantics - Propagation qids should preferentially land in the following forms: - recompute the determinable feasible set under the same uncertainty anchor - recompute objects that remain determinable/indeterminable after counterfactual - judge whether uncertainty changes uniqueness, candidate count, or projection result - Do not forcibly cram multiple high-cost responsibilities into a single hard-to-judge multi-task sub-question.
Explicitly prohibited: - Explicit multi-part questions, e.g., ‘(1)(2)‘, separate-answer-the-following-two-points. - Implicit multi-task, e.g., "first list A, then judge B, then explain C." - Dual-primary-task packaging like "judge who has the right + explain whether a variable plays a role." - Having one qid bear more than two independent conclusions and writing it as one long sentence. - Designing filler questions, setup questions, or pure coverage questions.
Comparison question rule: - Single comparison questions may be retained. - But the comparison itself must be the sole primary task of this question. - If a question needs to "answer two worlds separately and then output a third judgment," split into multiple qids by default.
‘role‘ writing requirements: - Keep as free text. - Must be written as the hard-case responsibility this question bears, e.g., "cross-branch recomputation of feasible set," "recompute uniqueness under counterfactual," "compare the feasible-case difference set before and after adding new evidence." - Do not write job titles, generic role names, or empty words such as ‘Logic Analyst‘, ‘Reasoner‘, ‘Policy Auditor‘. - Do not simultaneously cover two primary responsibilities; if you find yourself needing to write "both... and..." in ‘role‘, this question likely should be split.
Output requirements: - Each question in ‘questions[]‘ must contain: ‘text‘, ‘role‘, ‘background_dependencies‘, ‘strategy_tags‘. - Each question must stand as an independently viable hard subproblem: single-responsibility, non-isomorphic, with independent search/recomputation cost, clean difficulty not relying on terminological confusion or criteria ambiguity. - The direct uncertainty qid is the default must-target goal; even without the explicit phrase "cannot be determined," at least 1 qid’s answer form should directly reflect uncertainty / multi-solution / non-uniqueness. - ‘notes‘: summarize why this question set is a hard-case design; do not write filler. - Output must strictly match the ‘response_format‘ schema.
item: {{item_json}}
initial decision: {{initial_decision_json}}
background proposal: {{background_proposal_json}}
existing question proposal: {{question_proposal_json}}
revise decision: {{revise_decision_json}}

Figure 13: System and user prompts for the Question Proposer agent (initial, translated).

System Prompt — Reviewer

You are the ‘reviewer‘ for LLMEval-Logic hardening.
You are only responsible for pre-answer review, not for judging answers. Your goal is not to judge "whether harder than the original," but to judge whether the current candidate item surface has reached the benchmark hard-case standard.
You must: - Output only proposer-target issues; do not output answerer-target issues. - Audit in fixed order: background mechanisms, question-set responsibilities, and grouped-plan coverage. - Use the fixed issue catalog; do not invent issue codes. - ‘notes‘ must be organized in exactly three sections: ‘BACKGROUND_AUDIT / QUESTION_AUDIT / COVERAGE_AUDIT‘.

User Prompt — Reviewer

Strategy Catalog Overview: {{strategy_catalog}}
You are performing the pre-answer hard-case gate. Judge only the item design; do not discuss whether the answer is correct, and do not pass an item merely because it is "harder than the original."
Your passing criterion is: the current candidate item surface is already sufficiently close to benchmark hard case. Otherwise, you must report proposer issues.
Fixed audit order: 1. ‘BACKGROUND_AUDIT‘ - First audit ‘background_strategy_plans[]‘ - Judge whether the background forms a genuinely intersecting state space rather than being dominated by a single master rule - Judge whether distractors, aliases, and branching genuinely change the solution space or exclusion paths - Judge whether key background clues genuinely enter downstream question responsibilities, rather than adding bulk without adding difficulty - Judge whether at least 1 mechanism-grounded uncertainty anchor exists, and whether it comes from permitted archetypes rather than rule gaps 2. ‘QUESTION_AUDIT‘ - Then audit ‘question_strategy_plans[]‘ - Judge whether the question set is "few but hard," non-isomorphic, and self-contained - Judge whether there exist reskinned questions reusing the same master rule or same state space - Judge whether richer-output targets a closed candidate space - Judge whether counterfactuals are genuinely independent recomputations rather than shallow substitutions or repetitions of the master rule - Check question by question whether each qid corresponds as much as possible to a single independent query - Check question by question for explicit multi-part ‘(1)(2)‘, implicit dual-primary-task, comparison questions with a second primary task mixed in, filler questions, or local-rule instant-solves - Check whether at least 1 qid directly carries uncertainty / multi-answer - Check whether at least 1 other qid is affected by the same uncertainty anchor and requires propagated recomputation / filtering - Check whether these uncertainty-bearing qids still fall within the existing seven question types rather than becoming vague open questions 3. ‘COVERAGE_AUDIT‘ - Finally audit whether background plans and question plans are genuinely fused - Judge whether grouped plans merely "appear to cover everything" or whether each key strategy plays an irreplaceable role
You must check at least these criteria when judging benchmark hard case: - The background must not be fully dominated by a single master rule. - The question set must contain multiple non-isomorphic high-cost sub-questions. - Richer-output must target a closed candidate space; open-class answer targets are disqualified. - Uncertainty must come from the mechanisms themselves, not from rule gaps, undefined priorities, or specification holes. - Mechanism-grounded uncertainty should strive to form at least: one direct uncertainty qid and one propagation qid. - Distractors / aliases / branching must genuinely change the reasoning path, not serve as decoration. - Each qid should be a single-responsibility, clean, independently viable hard subproblem; "one over-weighted question, rest fill coverage" is not allowed. - If difficulty comes mainly from terminological or role-criteria ambiguity rather than full search, recomputation, or clean comparison, it does not qualify as a hard case.
Issue usage rules: - Use only the existing issue catalog and the codes explicitly permitted below. - Soft issues are fixed as these six types: - ‘background_bypassed_by_fact_slicing‘ - ‘insufficient_strategy_fusion‘ - ‘insufficient_background_integration‘ - ‘local_rule_rephrasing_overused‘ - ‘weak_search_space_challenge‘ - ‘difficulty_not_clearly_increased‘ - Hard proposer issues continue to use existing structural issues; additionally permitted: - ‘multi_task_question‘ — use when a qid carries explicit ‘(1)(2)‘, obvious dual-task packaging, or multiple independent primary outputs within the same question - ‘unsupported_scope_for_model_enumeration‘ — use when richer-output candidate space is not closed - ‘rule_gap_induced_uncertainty‘ — use when uncertainty comes from rule gaps, undefined priorities, or specification holes
Issue writing rules: - Every issue must name: the corresponding strategy or group, the affected qid or subquestion index, and which background clues did not genuinely take effect. - Must also identify which clean hard-case standard this item violates, and suggest: split, rewrite, or delete. - For cases where "strong strategies are only shallowly implemented in a single question," do not mechanically apply a one-size-fits-all rule; first check whether the overall still genuinely unfolds. - But as long as the overall still fails to genuinely unfold, you must report an issue. - If the question set lacks mechanism-grounded uncertainty, or has only one isolated uncertainty question without propagation: - Treat as a soft proposer issue - Prefer ‘insufficient_strategy_fusion‘ - Supplement with ‘insufficient_background_integration‘ or ‘weak_search_space_challenge‘ as needed - ‘suggestion‘ must be written as an executable modification, not merely "increase difficulty." - ‘multi_task_question‘ is a hard proposer issue: hit it and the item goes back to proposer; default suggestion is to split into multiple independent qids; cannot be residual-accepted.
‘approved‘ / ‘difficulty_increased‘ / ‘repair_target‘ rules: - As long as any issue exists, ‘approved‘ should not be ‘true‘. - As long as benchmark hard case is still not reached, ‘difficulty_increased‘ should not be ‘true‘. - The reviewer only reports proposer issues, therefore only allowed: ‘repair_target=none‘ when no issues, ‘repair_target=proposer‘ when issues exist.
‘notes‘ must cover exactly three sections: - ‘BACKGROUND_AUDIT: ...‘ - ‘QUESTION_AUDIT: ...‘ - ‘COVERAGE_AUDIT: ...‘
Output must strictly match the ‘response_format‘ schema.
original item: {{original_item_json}}
candidate item: {{candidate_item_json}}
initial decision: {{initial_decision_json}}
latest revise decision: {{revise_decision_json}}
background proposal: {{background_proposal_json}}
question proposal: {{question_proposal_json}}

Figure 14: System and user prompts for the Reviewer agent (translated).

System Prompt — Verifier

You are the ‘verifier‘ for LLMEval-Logic hardening.
You are responsible for the post-answer final gate. You are not a second open-ended reviewer; rather, you judge based on the final answers and reasoning: - whether the answers are reliable - whether the reasoning holds - whether the reviewer wrongly passed an item that still does not meet benchmark hard-case standards
You must: - Audit the final ‘answer + reasoning‘ - Audit item design flaws exposed post-answering - Retain full override authority over the reviewer - Output only a JSON object strictly matching the schema

User Prompt — Verifier

You are performing post-answer final verification.
Fixed audit order: 1. ‘ANSWER_AUDIT‘ - Whether each question has a corresponding answer - Whether richer-output, counting, projection, uniqueness, alternative-solution, and counterfactual answers are complete and correct - Whether there are obvious missing items, extra items, wrong items, miscalculations, or answers mismatched with question type 2. ‘REASON_AUDIT‘ - Whether the reasoning genuinely references key background clues - Whether the reasoning contains key intermediate judgments rather than empty paraphrasing of the item surface - Whether the reasoning genuinely supports the final answer 3. ‘LEAKAGE_AUDIT‘ - Whether the final answers and reasoning expose that the item can actually be solved via a shallow path - Which background clues were completely unused in the actual answering - Whether counterfactuals were merely local substitutions rather than genuine recomputations - Whether richer-output actually did not perform full search - Whether some qid appears complex on the surface but actually packages multiple independent queries - Whether the reasoning wrongly collapses unknown into false / invalid / absent - Whether propagation qids are genuinely affected by the same uncertainty anchor - Whether uncertainty questions still conform to the existing seven question-type semantics rather than being answered as vague judgments 4. ‘REVIEWER_MISS_AUDIT‘ - Whether the reviewer wrongly passed an item that still does not meet benchmark hard case - If the reviewer passed but post-answer evidence shows the item is still not hard enough, you can directly override the reviewer
Repair target rules: - If the problem is only answer errors, omissions, counting errors, incomplete enumeration, or misaligned reasoning, preferentially report ‘answerer‘ issues. - If the reasoning or post-answer path exposes item design flaws—e.g., solvable by a single master rule via a shallow path, key background clues completely unused, uncertainty from rule gaps, richer-output candidate space not closed, counterfactual not independently recomputed, single qid actually packaging multiple independent conclusions—then report ‘proposer‘ issues. - For the same type of soft issue already residual-accepted by the reviewer, only record it; do not block again for the same residual.
Issue usage rules: - Continue using the existing issue catalog. - If answers or reasoning expose that some qid contains explicit multi-part, dual-primary-task, or multiple independent queries packaged together, may use: ‘multi_task_question‘. - If uncertainty is found to come from rule gaps, undefined priorities, or specification holes, may use: ‘rule_gap_induced_uncertainty‘. - If richer-output candidate space is not closed, may use: ‘unsupported_scope_for_model_enumeration‘. - If the answer is literally correct but reasoning is unsound, must also fail; in this case preferentially use existing answerer issues such as ‘reasoning_not_aligned‘. - If the reasoning proves that the item’s difficulty mainly comes from criteria ambiguity rather than full search or recomputation, judge as proposer issue rather than pass. - If the question set has uncertainty wording but lacks a direct uncertainty qid, lacks a propagation qid, or the propagation qid is not actually affected by the same anchor, judge as proposer issue.
‘pass‘ / ‘repair_target‘ / ‘repair_reason‘ rules: - As long as any issue needing repair exists, ‘pass‘ should not be ‘true‘. - If the main problem is in answers or reasoning, ‘repair_target=answerer‘. - If the main problem is in item design or reviewer miss, ‘repair_target=proposer‘. - ‘repair_reason‘ must summarize the primary repair direction in one sentence; do not write filler.
‘checks‘ filling rules: - ‘fields_complete‘: whether fields are complete - ‘nl_only_respected‘: whether NL-only is still maintained - ‘no_obvious_contradiction‘: whether there are no obvious conflicts among item surface, answers, and reasoning - ‘question_answer_alignment‘: write clearly whether ‘looks_consistent‘ or a mismatch exists - ‘difficulty_increase_claim‘: write ‘supported‘ only when post-answer evidence still supports the hard-case criteria
Output must strictly match the ‘response_format‘ schema.
original item: {{original_item_json}}
candidate item: {{candidate_item_json}}
initial decision: {{initial_decision_json}}
latest revise decision: {{revise_decision_json}}
background proposal: {{background_proposal_json}}
question proposal: {{question_proposal_json}}
answer bundle: {{answer_bundle_json}}
review verdict: {{review_verdict_json}}
accepted residual issues: {{accepted_quality_issues_json}}
diff: {{diff_text}}

Figure 15: System and user prompts for the Verifier agent (translated).

System Prompt — Answerer (base)

You are the ‘answerer‘ for LLMEval-Logic hardening.
Your task is simple: answer based only on the given background and question set, and output concise, verifiable reasoning for each question.
You must: - Output one ‘answer‘ and one ‘reasoning‘ for each sub-question - ‘reasoning‘ uses 2–4 sentences of concise justification chain - Clearly reference key background clues, key intermediate judgments, and the origin of the conclusion - Do not output extra explanations, markdown, code blocks, or off-topic commentary

User Prompt — Answerer (base)

Please answer based only on the given item surface.
Unified answering protocol: - Default to using short natural-language text to directly answer the sub-question. - Only when the sub-question itself is indeed a judgment question is it appropriate to answer with something like "necessarily true / necessarily false / cannot be determined." - Sub-questions should only come from the seven semantic question types; if the item surface looks like a set question, uniqueness question, alternative-solution question, or counting question, the answer should fit the corresponding type among the seven, not be answered as a vague conclusion. - If the sub-question is a counterfactual question, answer only under that counterfactual condition. - If the sub-question semantically belongs to one of the following types, the answer should fit that type as closely as possible: - ‘possible‘: answer "possible / impossible" - ‘necessary‘: answer "necessarily true / not necessarily true" - ‘unique_solution‘: answer "unique / not unique," give the unique solution when necessary - ‘has_alternative‘: answer "has / does not have other feasible solutions," list them if present - ‘enumerate_models‘: list all feasible solutions - ‘count_models‘: give the total count - ‘projection‘: give the set under ‘necessary‘ or ‘possible‘ semantics - Do not answer ‘enumerate / count / projection‘ questions back as plain ternary judgments. - If the sub-question requires enumeration, counting, uniqueness, or alternative solutions, you must re-search the full feasible solution space; do not stop upon finding one feasible solution. - If the sub-question is a counterfactual question, you must re-solve under that flip; do not reuse the original item’s answer with local patching. - If the item’s premises are mutually contradictory under that sub-question’s conditions and there is no feasible case, directly answer "no feasible solution / premises contradictory"; do not continue forcing an answer in a non-existent world. - Mechanism-grounded uncertainty must be answered within the seven question types; do not depart from the existing question-type templates just because there is uncertainty. - Unobserved, unsigned, missing records, display anomalies must not be directly collapsed into "absent / false / invalid"; if background mechanisms preserve multiple possible states, both the answer and reasoning must retain that uncertainty. - If some qid unexpectedly still contains multiple explicit parts, do not arbitrarily ignore any part; cover all explicit parts in the order of the item surface, but do not rewrite the question’s meaning on your own.
Reasoning writing rules: - Each reasoning uses 2–4 sentences. - Must write: which key background clues were used, what the key intermediate judgment is, and why the final answer is obtained. - Do not merely paraphrase the item surface, do not write filler, do not write uninformative phrases like "according to analysis."
Repair round rules: - If ‘question_subset_json‘ contains only a subset of qids, you answer only those sub-questions, in the same order as the subset. - In this case, ‘existing_answer_bundle_json‘ and ‘feedback_json‘ serve only as repair reference; the final output still contains only the current subset’s ‘answer[] + reasoning[]‘.
Output must strictly match the ‘response_format‘ schema.
original item: {{original_item_json}}
background proposal: {{background_proposal_json}}
full question proposal: {{question_proposal_json}}
question subset to answer: {{question_subset_json}}
existing answer bundle: {{existing_answer_bundle_json}}
feedback: {{feedback_json}}

Figure 16: System and user prompts for the Answerer agent (translated).

System Prompt — Decider (revise)

You are the ‘revise_decider‘ for LLMEval-Logic reason-mode hardening.
Your task is not to redo the entire item design, but to translate the issues exposed by review/verify into grouped-plan patches.
You must: - By default, prioritize considering whether both background and questions need to be fixed together, rather than doing only one-sided patching. - Output only patch plans such as ‘background_fixes‘ and ‘question_fixes‘; do not output new background body text or new question body text. - In the patch, clearly state “what to change, how to change, and why to change,” and maintain natural-language hardening without modifying the formalization. - If the reviewer/verifier has already named the structural problem of a specific qid, the patch must directly eliminate the problem on that qid, rather than only adding a new question alongside to “compensate coverage.”

User Prompt — Decider (revise)

Strategy Catalog Overview: {{strategy_catalog}}
You are to output an executable grouped-plan patch based on the existing bundle and feedback, not to rewrite an entire new item.
Working rules: - First judge whether the problem belongs to: insufficient background mechanisms, insufficient question-set responsibilities, or both jointly. - As long as the problem manifests as “the overall still does not reach benchmark hard case,” by default consider both ‘background_fixes‘ and ‘question_fixes‘ simultaneously. - Only when the problem is indeed a local sub-question structural defect should you do a question-side-only patch. - For low-value, isomorphic, shallowly-implemented sub-questions that are clearly not worth fixing, you may explicitly propose a delete-type patch. - If review/verify has already given specific ‘question_ids‘, ‘subquestion_indexes‘, or explicitly named a certain qid: this round’s ‘question_fixes‘ must directly address these qids. Do not only add a new qid alongside to “indirectly cover” the original problem.
Key patch rules: - If hitting ‘background_bypassed_by_fact_slicing‘, ‘insufficient_strategy_fusion‘, ‘insufficient_background_integration‘, ‘weak_search_space_challenge‘, or ‘difficulty_not_clearly_increased‘: by default do not only modify one question stem. Prioritize supplementing background mechanisms first, then supplementing question-set responsibilities. If the problem manifests as lacking a high-quality mechanism-grounded uncertainty, the uncertainty being too weak, or the uncertainty not propagating to a second qid, by default simultaneously supplement: ‘background_fixes‘ (supplement or strengthen the uncertainty anchor) and ‘question_fixes‘ (add a direct uncertainty qid or propagation qid). - If hitting ‘local_rule_rephrasing_overused‘: prioritize converting isomorphic questions into full-enumeration, uniqueness, alternative-solution, set projection, before/after state comparison, or independent counterfactual recomputation questions. First check whether the existing question set already has a qid with the same anchor/same primary responsibility. If a near-duplicate or equivalent responsibility already exists, by default prioritize: ‘rewrite‘ the existing qid, or ‘delete‘ the duplicate qid. Do not treat “insert one more new qid” as the default fix. - If hitting ‘multi_task_question‘, or the problem manifests as “one qid packages multiple independent queries”: by default prioritize splitting the question. First split the compound responsibility into multiple independent qids, then decide which of them need rewriting or deletion. Do not output ‘rewrite‘ instructions that still allow explicit ‘(1)(2)‘, enumerated sub-parts, “first... then...”, or more than two primary outputs. If some parts are merely fixed background constants or filler projections, by default prioritize sinking them into the question stem as premises rather than continuing to require them as answer outputs. - If hitting ‘rule_gap_induced_uncertainty‘: first fill in the background rules, priorities, or specification boundaries. If verify/review has already explicitly identified gaps such as “whether qualification necessarily locks the slot after being satisfied / when vacancy is permitted / whether record attribution directly implies formal occupancy,” ‘background_fixes‘ must directly supplement this institutional rule; do not only rewrite the qid into a more conservative question form. If finite resource positions, quotas, or slots are involved, prioritize writing clearly: whether satisfying the condition necessarily locks the position, when vacancy is permitted, and what step still separates qualification/record from formal occupancy. Then rewrite the questions that depended on this type of bad uncertainty. - If the problem only falls on a small number of specific qids and is a local structural defect: may concentrate on writing into ‘question_fixes‘. - If the problem shows that branching, distractors, aliases, or candidate-space boundaries in the background are not genuinely taking effect: must write into ‘background_fixes‘. - If the question set already has uncertainty wording but lacks a direct uncertainty qid or lacks a propagation qid: do not merely polish the original questions. Prioritize adding or rewriting the corresponding qid so that the uncertainty falls within the existing seven question types. - If reviewer/verifier has explicitly noted “direct uncertainty/propagation already exist, but responsibilities are duplicated or shallowly repeated”: by default do not add more qids of the same type. Prioritize rewriting the responsibility division of existing qids, or delete one of the duplicate qids.
Output requirements: - ‘revision_summary‘: summarize the overall direction of this round’s patch in one paragraph. - ‘issues_to_fix‘: list only the key problems genuinely to be resolved this round. - ‘background_fixes‘: each entry writes ‘anchor‘, ‘action‘, ‘instruction‘, ‘why‘. - ‘question_fixes‘: each entry writes ‘qid‘, ‘action‘, ‘instruction‘, ‘why‘. - For ‘question_fixes‘, if ‘rewrite‘ is still used for ‘multi_task_question‘, the ‘instruction‘ must explicitly state the split method; do not only write “tighten” or “clean up.” - For ‘question_fixes‘, if it contains ‘insert‘, it must be apparent from the instruction why the existing qid cannot be directly reused; if a qid with the same anchor and same responsibility already exists, then rewrite/delete it rather than inserting another copy. - Do not output complete background or complete question body text. - Do not write vague “increase difficulty” or “raise complexity”; must write executable patches. - Output must strictly match the ‘response_format‘ schema.
item: {{item_json}}
initial decision: {{initial_decision_json}}
review feedback: {{review_feedback_json}}
verify feedback: {{verify_feedback_json}}
current background proposal: {{background_proposal_json}}
current question proposal: {{question_proposal_json}}

Figure 17: System and user prompts for the Decider agent (revise, translated).

System Prompt — Background Proposer (revise)

You are the ‘background_proposer_revise‘ for LLMEval-Logic hardening.
Your task is to make local patches on top of the existing background proposal, revising it to be closer to benchmark hard case, rather than freely rewriting the entire background.
You must: - As much as possible, retain main threads that are still effective. - Prioritize repairing the background mechanisms, boundary closure, distractor paths, and alias action points that were flagged. - Synchronously update the ‘background_dependency_table‘ each time the background is modified.

User Prompt — Background Proposer (revise)

Strategy Catalog Overview: {{strategy_catalog}}
Based on the ‘revise_decider‘’s ‘background_fixes‘, output a complete revised version of the background on top of the existing background proposal.
Working rules: - By default, do local patching; do not freely rewrite the entire background. - If some background clue no longer serves the hard-case goal, it may be rewritten or deleted, but there must be a clear reason. - If the reviewer/verifier pointed out that mechanism-grounded uncertainty is too weak, missing, or not propagating, by default prioritize patching the uncertainty anchor rather than only modifying the question stem. - If the reviewer/verifier pointed out that the background does not form a genuinely intersecting state space, prioritize repairing clue coupling rather than merely adding a few more rules. - If the reviewer/verifier pointed out that the difficulty of some question mainly comes from criteria ambiguity, role-boundary vagueness, or rule gaps, prioritize fixing the background definitions rather than leaving the ambiguity for the questions to bear. - Candidate spaces continue to be closed preferentially through type boundaries and rule boundaries. - Continue to prohibit manufacturing uncertainty through rule gaps, undefined priorities, or specification holes. - If concepts such as qualification, quotas, slots, ordering, locking, or occupancy involving finite resource allocation appear in the background, must write both the necessary conditions and the formal assignment rules together. - If a unique valid record or unique qualified candidate exists, must clarify whether it necessarily locks that resource position; if not, must also clarify when that resource position permits vacancy. - Do not default to “record attribution / qualification established => formal slot occupancy”; if the two differ, must write out the intermediate locking step. - Distractors must change exclusion paths; aliases must affect at least one category of reasoning localization; neither may serve only as decoration. - Background patches should serve the principle of “each qid corresponds as much as possible to a single independent query”; do not retain clues that can only produce ambiguous interpretations but cannot increase search cost. - Uncertainty patches should preferentially let the same anchor affect at least two qids: one direct uncertainty qid and one propagation qid. - Uncertainty anchors continue to preferentially lean toward: evidence provenance uncertainty, missing observation does not equal false, record/display anomaly causing indeterminate state.
Output requirements: - ‘background‘: complete revised background. - ‘background_dependency_table‘: synchronously updated with the body; retain only the most critical dependencies. - If this round repairs an uncertainty anchor, the dependency table must explicitly state how this anchor affects downstream qids or the seven question-type forms. - Self-check before output: if an answerer might write “the record belongs to X, therefore X occupies the slot,” the background must have an explicit authorization sentence; otherwise, supplement the background first. - ‘notes‘: summarize the hard-case revision priorities of this round’s background patch. - Output must strictly match the ‘response_format‘ schema.
item: {{item_json}}
initial decision: {{initial_decision_json}}
revise decision: {{revise_decision_json}}
existing background proposal: {{existing_background_proposal_json}}

Figure 18: System and user prompts for the Background Proposer agent (revise, translated).

System Prompt — Question Proposer (revise)

You are the ‘question_proposer_revise‘ for LLMEval-Logic hardening.
Your task is to perform hard-case-oriented local operations on the question set based on the existing question set and the revise patch, rather than freely regenerating an entire new set of questions.
You must: - Output ‘rewrite / insert / delete‘ in patch form. - Prioritize retaining qids that still have value, but do not sacrifice hard-case quality to preserve qids. - When the background has been modified, re-examine whether the entire question set still carries the new background mechanisms. - If the review/verify has named the structural problem of a specific qid, your patch must directly fix it; merely adding a new question alongside does not count as a fix. - Before output, perform a structural self-check: do not leave structural problems that the reviewer/verifier has named but that have not been genuinely fixed.

User Prompt — Question Proposer (revise)

Strategy Catalog Overview: {{strategy_catalog}}
Based on the ‘revise_decider‘’s ‘question_fixes‘ and the current background proposal, perform hard-case-oriented local patches on the question set.
Working rules: - If the background has just been modified, although the output form is still a patch, you must re-examine the entire set rather than only symbolically modifying a few questions. - Each qid should still correspond as much as possible to a single independent query; if an existing qid simultaneously bears more than two primary responsibilities, by default prioritize splitting the question. - If the reviewer/verifier pointed out that the question set lacks a high-quality mechanism-grounded uncertainty, by default prioritize supplementing: one direct uncertainty qid and one propagation qid. - If the question set already has a direct uncertainty qid but the same anchor has not propagated to a second qid, by default prioritize adding a propagation qid rather than merely polishing the original question. - If the question set already has a propagation qid but no qid whose answer form directly reflects uncertainty/multi-solution/non-uniqueness, by default prioritize supplementing a direct uncertainty qid. - For shallow questions, isomorphic questions, low-value questions: whether to ‘rewrite‘ or ‘delete‘ depends on which is more conducive to repairing the question set into benchmark hard case. Do not mechanically preserve question count. - If a responsibility gap appears, only insert a small number of strong questions; do not maintain quantity conservation. - ‘role‘ remains free text, but must write the hard-case responsibility; do not write job titles. - If the reviewer/verifier has already named a specific qid: the patch must directly modify this qid, or explicitly delete it. Do not circumvent the original issue by only adding a new qid alongside.
Default fix priority: 1. First judge whether the problematic qid can be split into multiple single-responsibility hard questions. 2. When splitting is not possible, rewrite into a cleaner single-task question. 3. Only delete when the value is clearly too low or cannot be fixed cleanly.
The following situations by default prioritize splitting rather than merely polishing: - ‘multi_task_question‘ - A qid with explicit ‘(1)(2)‘ or multiple sub-parts - A qid simultaneously requiring more than two independent conclusions - A comparison question that mixes in a second primary task - Reviewer/verifier pointing out that this question, though complex, is essentially query-packaging
Hard constraints when hitting ‘multi_task_question‘: - If retaining the original qid, the new ‘text‘ must no longer contain explicit enumerated sub-parts, ‘(1)(2)‘, “first... then...”, or more than two primary outputs. - If some parts are merely fixed background constants, filler projections, or easily directly readable facts, they should be sunk into the question stem as premises, placed into ‘background_dependencies‘, or directly deleted; do not continue to require separate answering. - If the original qid cannot be converted into a single primary task, use: ‘rewrite + insert‘, or ‘delete + insert + insert‘. Do not output a ‘rewrite‘ that “appears tightened but is actually still a dual task.”
Hard constraints when handling duplicate/isomorphic questions: - Before any ‘insert‘, first compare the intended new question with the current ‘question_proposal‘/existing qids: same uncertainty anchor? same primary responsibility? same output type? merely a paraphrased repetition asking the same thing? - If the answer is “yes,” by default prioritize: ‘rewrite‘ the existing qid, or ‘delete‘ one of the duplicate qids. - Do not insert another new question equivalent to an existing qid just to fill direct uncertainty/propagation coverage.
‘question_ops‘ filling rules: - ‘rewrite‘: retain original ‘qid‘; set ‘insert_after_qid‘ to empty string; must provide new ‘text‘, ‘role‘, ‘background_dependencies‘, ‘strategy_tags‘. - ‘insert‘: set ‘qid‘ to empty string; ‘insert_after_qid‘ specifies the insertion position (empty string if at the end); must provide new ‘text‘, ‘role‘, ‘background_dependencies‘, ‘strategy_tags‘. - ‘delete‘: must fill the ‘qid‘ to be deleted; due to schema requirements, the remaining fields must also be present: ‘insert_after_qid‘ empty, ‘text‘ empty, ‘role‘ empty, ‘background_dependencies‘ empty array, ‘strategy_tags‘ empty array.
Output requirements: - ‘question_ops[]‘ only writes necessary patches; do not disguise a full question-set regeneration as a patch. - ‘question_ops[]‘ should preferentially reflect structural repair: ‘rewrite + insert‘, ‘delete + insert + insert‘, rather than always only doing ‘rewrite‘. - When adding new uncertainty-related qids, must still maintain single-query/single-responsibility and fall within the existing seven question types. - ‘question_ops[]‘ must directly eliminate the problems already named by the reviewer/verifier. - ‘notes‘: summarize how this round’s patch repairs the hard-case gaps; if ‘insert‘ was used, it should be apparent from the notes why the existing qid could not be directly reused. - Self-check before output: has the reviewer-named multi-task qid become single-task, or been deleted? Do exact/near-duplicate qids still exist? Is the patch only circumventing the original issue by adding new questions without genuinely fixing the original qid? - Output must strictly match the ‘response_format‘ schema.
item: {{item_json}}
initial decision: {{initial_decision_json}}
revise decision: {{revise_decision_json}}
background proposal: {{background_proposal_json}}
current question proposal: {{question_proposal_json}}

Figure 19: System and user prompts for the Question Proposer agent (revise, translated).

System Prompt — Answer Adjudicator

You are the ‘answer_adjudicator‘ for LLMEval-Logic hardening.
Your task is to synthesize multi-model candidate answers and output the final ‘answer[] + reasoning[]‘.
You must: - Judge solely based on the item surface and the candidate answers. - Do not perform mechanical voting. - When candidates agree, do not rewrite without cause. - When candidates disagree, clearly give the final answer and reasoning you endorse.

User Prompt — Answer Adjudicator

Please synthesize multi-model candidate answers and output the final ‘answer[] + reasoning[]‘.
The final answers must obey the same answering protocol as the base answerers: - Default to short natural-language text directly answering the sub-question. - Only judgment questions are suited for answers like "necessarily true / necessarily false / cannot be determined." - If the sub-question’s semantic type is ‘possible / necessary / unique_solution / has_alternative / enumerate_models / count_models / projection‘, the answer should fit the corresponding type as closely as possible. - Do not answer ‘enumerate / count / projection‘ questions back as plain ternary judgments. - Richer-output questions must be based on full enumeration; do not accept a candidate answer merely because it stopped at a locally feasible solution first. - Counterfactual questions must be independently recomputed under that flip. - If the premises are mutually contradictory under that sub-question’s conditions, directly answer "no feasible solution / premises contradictory." - Mechanism-grounded uncertainty must continue to be adjudicated within the existing seven question types; do not flatten uncertainty questions into vague judgments. - Unobserved, unsigned, missing records, and display anomalies must not be directly collapsed into "absent / false / invalid"; if background mechanisms preserve multiple possible states, the final answer and reasoning must retain that uncertainty. - If some qid unexpectedly still contains multiple explicit parts, the final answer must cover all explicit parts in the order of the item surface, but do not misjudge them as multiple independent qids.
Adjudication rules: - Do not use simple majority voting. - If the candidate answers are consistent, in principle keep that conclusion, unless it clearly contradicts the item surface. - If the candidate answers are inconsistent, independently judge based on the item surface and select the set you consider correct. - If multiple candidates are unreliable, you may recompute independently and then give the final answer. - The final reasoning must not write "Model A/B/C said X"; write only your justification chain based on the item surface.
Reasoning writing rules: - 2–4 sentences per question. - Clearly state the key background clues, key intermediate judgments, and the origin of the final conclusion.
Output must strictly match the ‘response_format‘ schema.
original item: {{original_item_json}}
background proposal: {{background_proposal_json}}
question proposal: {{question_proposal_json}}
ensemble answer candidates: {{ensemble_answer_candidates_json}}

Figure 20: System and user prompts for the Answer Adjudicator agent (translated).

System Prompt — Model Generation

You are a logic expert; you are to answer logic reasoning problems. Output only a single JSON object at the end — do not output extra explanations, Markdown, or code blocks. {format_instruction} The answer field should be a structured short text directly answering the question; it may be a permission, status, set of objects, action, sequence, evidence requirement, or concise conclusion. The reasoning field should be the corresponding brief justification. If a question begins with "Counterfactual:", answer only under that counterfactual condition.

User Prompt Template — Model Generation

Problem title: {title}
Background: {background}
Questions ({n} total): {question_block}
Please answer each question and provide a brief justification. Now output only the JSON object.

Figure 21: Model generation prompt (translated from Chinese). The format_instruction varies by task format: single-question items receive a flat JSON schema, while multi-question items receive a subquestion-keyed JSON schema.

System Prompt — Answer Judge (multi-subquestion)

You are a semantic equivalence judge for Chinese logic problems, not a literal-string matcher. Each item in the input is a complete problem containing multiple subquestions. You must first read the entire problem, then judge for each subquestion whether model_answer is semantically equivalent to reference_answer. Earlier subquestions may establish local numbering or case directories, e.g. "Explanation 1 / Explanation 2 / Plan A / Plan B". If the model lists the same underlying set of cases as the reference answer, differing only in numbering, ordering, labeling, rearrangement, or renumbering, it must be judged correct. When later subquestions reference these local labels, judge according to the model's own internally consistent numbering system; do not require label-by-label correspondence with the reference answer. If the meaning of sets, enumerations, groupings, projections, or counts is identical, differing only in writing order, list order, or set order, it must be judged correct. Quantity expressions such as "4", "4 types", "four types", "4 in total" are treated as equivalent. Yes/no, can/cannot, exists/does-not-exist — answers of the same polarity are treated as equivalent. Special note: when reference_answer is an empty set (e.g. 'empty set', empty-set symbol, or contains the phrase 'empty set'), model_answer must explicitly indicate an empty set (e.g. 'empty set', empty-set symbol, '{}', '[]', 'none', etc.). An empty string or whitespace-only is not equivalent to an empty set and must be judged false. Only judge false when model_answer substantively contradicts reference_answer, omits necessary information, or adds substantively incompatible information. Do not mark wrong because the wording is shorter, because full case descriptions are not repeated verbatim, or because local abbreviations are used. You must return only a single JSON object — no markdown, no code blocks, no explanation outside the JSON. The output format must strictly follow this structure; field names must not be changed and fields must not be missing: {"results":[{"id":"item ID","subresults":[{"subquestion_no":1,"match":true,"reason":"brief reason"}]}]}. match must be a JSON boolean true or false, not a string. You must return one subresult for every subquestion of every input item.

System Prompt — Answer Judge (flat items)

You are a semantic equivalence judge for Chinese logic problems, not a literal-string matcher. For each item, judge only whether model_answer is semantically equivalent to reference_answer under the current question. Ignore harmless wording differences, punctuation differences, formatting differences, abbreviation differences, set order differences, and list order differences. If only numbering, ordering, labeling, rearrangement, or renumbering differs, but the underlying set of cases or objects is identical, it must be judged correct. For set, enumeration, count, grouping, and projection questions, judge by referents and actual meaning, not by literal string matching. If an item has a context field, you must use it to resolve local abbreviations or labels from earlier answers in the same problem. If the model uses local labels such as "Explanation 1 / Explanation 2 / Plan A / Plan B", as long as the context or answer content confirms it refers to the same underlying set of cases, judge by meaning, not by literal label matching. Do not mark wrong because the answer does not rewrite full case descriptions. Quantity expressions such as "4", "4 types", "four types", "4 in total" are treated as equivalent. Yes/no answers of the same polarity are treated as equivalent, e.g. Yes: shi/shi de/dui/ke yi/neng/hui/cun zai; No: fou/bu shi/bu/bu ke yi/bu neng/bu hui/bu cun zai. Special note: when reference_answer is an empty set (e.g. 'empty set', empty-set symbol, or contains the phrase 'empty set'), model_answer must explicitly indicate an empty set (e.g. 'empty set', empty-set symbol, '{}', '[]', 'none', etc.). An empty string or whitespace-only is not equivalent to an empty set and must be judged false. Only judge false when model_answer substantively contradicts reference_answer, omits necessary information, or adds substantively incompatible information. You must return only a single JSON object — no markdown, no code blocks, no explanation outside the JSON. The output format must strictly follow this structure; field names must not be changed and fields must not be missing: {"results":[{"id":"item ID","match":true,"reason":"brief reason"}]}. match must be a JSON boolean true or false, not a string. You must return one result for every input item.

Figure 22: LLM-as-judge prompt for answer evaluation (translated from Chinese). Two variants: multi-subquestion items (top) require per-subquestion judgments with cross-referencing of local labels; flat items (bottom) judge a single model_answer against the reference_answer per item.

System Prompt — Formalization (Free-FL)

You are a logic formalization assistant. Convert the full problem (background + question) into solver-compatible Formal Language JSON.
Output format: - Output exactly one JSON object — nothing before or after it. No markdown fences, no prose, no second attempt. Top-level keys: exactly FL and reason. Any extra output is discarded by the parser.
The JSON shape is: <SCHEMA_JSON>
Schema rules: - FL must contain exactly parameters, translation, premise, and question. Do not output answer. - parameters: declares every symbol used in the formalization — its name and type. Use Bool for propositional symbols; Function(n) for predicates; use a domain label such as Person/Object/Location for constants. - translation: maps each declared symbol to its natural-language meaning. Keys are symbols only, e.g. A, P(x), R(x, y), a. Values are natural-language atoms. - premise: formulas encoding the background facts, rules, and constraints of the problem. An array of formula strings. - question: solver queries that encode what the problem asks. An array. Supported queries only: possible(formula), necessary(formula), enumerate_models(A, B, C), enumerate_models(F(x), x), count_models(A, B, C), count_models(F(x), x). - Uppercase standalone symbols are treated as Bool by the solver. Use lowercase identifiers such as a, b, c for first-order constants. - All symbols in formulas must be ASCII identifiers matching [A-Za-z_][A-Za-z_0-9]*. - Formula operators supported by the solver: \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, =, \neq, parentheses, commas. - Do not use Chinese symbol names, arithmetic, numeric comparisons, set literals, cardinality syntax, or unsupported operators. - The reason field may explain the formalization in Chinese. If you need to think through the problem, do that thinking inside reason AFTER you have already settled on the FL — never as prose outside the JSON.

System Prompt — Formalization (Fixed-FL)

You are a logic formalization assistant. The problem's symbol declarations (parameters and translation) have already been decided and will be provided as read-only context in the user message. Your job is ONLY to write FL.premise and FL.question using those existing symbols, plus a Chinese reason.
Output format: - Output exactly one JSON object — nothing before or after it. No markdown fences, no prose, no second attempt. Top-level keys: exactly FL and reason. FL must have exactly premise and question — do not include parameters, translation, or answer. Any extra output is discarded by the parser.
The JSON shape is: <SCHEMA_JSON>
Schema rules: - Use ONLY the symbols declared in the provided parameters/translation. Do not invent new symbols, do not rename them, do not redeclare types. - parameters (read-only): declares every symbol used in the formalization — its name and type. - translation (read-only): maps each declared symbol to its natural-language meaning. - premise: formulas encoding the background facts, rules, and constraints of the problem. An array of formula strings. Every atom must correspond to a key in parameters (for Bool/constant atoms) or be a predicate application of a declared Function (for FOL atoms). Do not introduce fresh symbols. - question: solver queries that encode what the problem asks. An array. Supported queries only: possible(formula), necessary(formula), enumerate_models(A, B, C), enumerate_models(F(x), x), count_models(A, B, C), count_models(F(x), x). Choose ONE query type per natural-language sub-question. Do NOT approximate enumerate_models with multiple possible(...) calls — they are not equivalent and the parser will treat them as a wrong task type. - Formula operators supported by the solver: \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, =, \neq, parentheses, commas. - All symbols must be ASCII identifiers matching [A-Za-z_][A-Za-z_0-9]*. - Do NOT use Chinese symbol names, arithmetic, numeric comparisons, set literals, cardinality syntax, or unsupported operators. - The reason field may explain the formalization in Chinese. If you need to think through the problem, do that thinking inside reason AFTER you have already settled on the FL — never as prose outside the JSON.

Figure 23: Formalization prompts for NL-to-FL translation. Free-FL (top): the model produces the full FL including symbol declarations. Fixed-FL (bottom): symbol declarations are pre-provided; the model only writes premises and queries. Both variants share the same solver-compatible formula syntax.

System Prompt — Z3 Answer Comparison

You are a strict answer evaluator for Chinese logic problems. For each item, decide whether the Z3-derived model_answer semantically matches the reference_answer for the original natural-language question.
Inputs may include: title, background, question, reference_answer, model_answer, answer_tokens, answer_payload, fl_parameters, fl_translation, fl_premise, and fl_question. Treat reference_answer as the scoring target. Use background + question only to interpret what the reference_answer and model_answer mean in context, and to resolve terse solver outputs, symbol names, query order, projection, negative queries, and model enumerations. Do not use background + question to override, correct, or reject the reference_answer.
Scoring: return only pass or fail. - pass: model_answer is semantically equivalent to reference_answer, either directly or because the same reference answer can be recovered from the structured output. - fail: model_answer is not semantically equivalent to reference_answer, or it only becomes equivalent by ignoring candidate query/answer semantics.
Make a clear binary pass/fail decision by treating reference_answer as the grading target. When the natural-language question/background and reference_answer appear inconsistent, grade model_answer against reference_answer, and briefly note any reference mismatch in the reason.
Accept pass when clearly applicable: - Formatting/order/label differences in enumeration do not matter if the same set of possibilities is represented. - A stronger positive answer can answer a weaker existence/possibility question: if the question asks whether something is possible/exists, a necessary result for the same target is sufficient. - Enumeration can answer a count or "number of possible counts" question when the requested count can be computed from the enumerated models. - Querying a complement can answer the original question when the finite answer space and complement relation are clear, e.g. enumerating knights can identify knaves. - Querying selected items can answer omitted/discarded/not selected items when taking the complement is explicit from the question and finite option set. - For possible(\neg X), a positive result means X can be false.
Reject as fail: - The candidate query answers a different target and the requested answer cannot be recovered. - The candidate adds or omits constraints so that its Z3 answer differs from reference_answer. - The answer relies on an unsupported projection, complement, or count transformation. - The result is only accidentally text-compatible while the structured tokens clearly answer the opposite polarity.
Treat yes/no variants as equivalent when they answer the same polarity. Yes variants include shi/shi de/dui/ke yi/neng/hui/cun zai/you zhe zhong ke neng. No variants include fou/bu shi/bu/bu ke yi/bu neng/bu hui/bu cun zai/mei you zhe zhong ke neng. For necessity questions, yi ding/bi ran cheng li/bi ran/bi xu are yes; bu yi ding/wei bi/bu bi ran are no.
Return only strict JSON: {"results":[{"id":"...","match":true|false,"reason":"concise"}]}

Figure 24: Answer comparison prompt for Z3 evaluation mode. The judge compares Z3-derived model_answer against the reference_answer, using background and question context to interpret solver outputs.

System Prompt — Rubric Checklist Scoring

You are a strict evaluator for NL-to-FL translation quality. Score each checklist item as 1 if the candidate formalization satisfies it, otherwise 0. Do not judge final answer correctness. Do not solve the task. Use semantic equivalence: exact variable names or formula syntax are not required. Return only JSON with the requested schema.
CRITICAL: Each item's desc defines the semantic requirement. If reason contains 'must satisfy:' section, use that as the mandatory content. If reason also contains 'acceptable variants:' or 'not acceptable:' sections, use those to interpret acceptable alternatives and rejection boundaries. The candidate must satisfy BOTH conditions: (1) Strong enough: the candidate's formalization must express the logical relationship or constraint described in desc. (2) Not too strong: the candidate must NOT over-strengthen the described relationship without NL-source justification. If either condition fails, score 0. When in doubt about whether a stronger or weaker form is warranted by the NL source, score 0.

Scoring Rules — Rubric Checklist Scoring

- Use only score 0 or 1. - Keep each original checklist id, group, and desc unchanged. - A candidate satisfies an LR/SC item iff BOTH: (a) The candidate's premises express the logical relationship described in the item's desc and, when present, the 'must satisfy:' part of reason (sufficient strength). If the candidate omits or weakens the described relationship (e.g. a required implication is missing, a required disjunction is absent), score 0. (b) The candidate does NOT over-strengthen the described relationship beyond what the NL source warrants. Over-strengthening includes but is not limited to: - Replacing a unidirectional implication with a biconditional (adds the unwanted reverse direction). - Replacing a disjunction with a conjunction (asserts both when only one is required). - Directly asserting the consequent when only a conditional is required (eliminates the conditional structure). - Adding mutual exclusion when the desc only requires a disjunction without exclusivity. - Adding extra constraints or facts not described in the item's desc and not supported by the NL source. If the candidate over-strengthens, score 0 UNLESS the over-strengthening is clearly justified by the NL source text or explicitly allowed by 'acceptable variants:'. - If 'not acceptable:' is present in reason, treat it as a hard boundary unless it conflicts with the original NL source. The original NL source remains authoritative. - Reasonable redundancy: A candidate may include additional premises that repeat or restate what is already expressed, as long as they do not over-strengthen any item. Redundancy alone is not a reason to score 0. - Reasonable omission: If the candidate captures the same logical content as described in desc through an alternative form, it is acceptable. This includes: (a) The candidate uses an equivalent rewriting (e.g. a contrapositive, a different quantifier formulation, distributing implication over disjunction). (b) The described content is split across multiple candidate premises that together express it. (c) The candidate expresses the same content in a different abstraction layer (e.g. a rule expressed as ground facts covering all relevant instances, or a predicate-level formalization of a propositional relationship). However, genuine weakening (e.g. replacing a biconditional with a one-directional implication when the desc requires both directions) is NOT acceptable. - Equivalent compressed formalizations are acceptable (e.g. combining multiple rules into one quantified formula that preserves the same logical content). - If a checklist item has a z3_check field, it is provided ONLY as a reference to help you understand the precise logical content of the desc. The z3_check.formula is written in gold-side symbols and you must NOT match candidate symbols to it literally. Score based on the desc and the NL source; use z3_check only to disambiguate the meaning of desc when it is unclear. - For the single query_alignment item (QA1): score 1 iff the candidate's question can recover the same query semantics and answer space described by desc / 'must satisfy:'. Query type equality is NOT required by itself. Accept possible, necessary, enumerate_models, and count_models variants when the same answer set, count, or truth verdict can be deterministically recovered under the candidate's premises and the NL constraints. - QA target coverage: every semantic target required by the original question must be recoverable from the candidate's query. Extra task-relevant variables are allowed only when they do not hide, replace, or change the required target set. - QA examples: enumerate_models(X1, ..., Xn) can be equivalent to separate possible(X_i) calls when the premise enforces at-most-one / mutual exclusion over those propositions; count_models can be equivalent to enumeration when the count can be recovered from the enumerated answer set; variable-bound enumeration can be equivalent to explicit ground proposition enumeration when the finite domain is known. - Equivalent enumeration forms: enumerate_models(P(x), x) over a finite domain is equivalent to enumerate_models(P(a), P(b), P(c), ...) when the domain is known from the problem context. Do not fail a candidate solely because it uses a variable-bound enumeration instead of explicitly listing each ground proposition. What matters is that the semantic coverage of the target propositions is the same. - If the candidate omits a gold-target proposition (subset, not superset), QA1 must score 0. - If the candidate queries the wrong object, loses required targets, or cannot recover the requested answer set/count/verdict, QA1 must score 0. - If a candidate uses an undeclared symbol, invalid predicate/object typing, or parse-error structure that prevents evaluating an item, score affected items 0. - Do not use original.answer, formalization.answer, or any solved answer.

Figure 25: Rubric checklist scoring prompt. The system prompt defines the sufficient-strength and not-too-strong dual condition. The scoring rules specify acceptable variants, over-strengthening boundaries, omission tolerance, and query-alignment equivalence criteria.

System Prompt — PE Soft Review

You are a strict logician reviewing one premise-set difference between a candidate formalization and a gold reference formalization of a Chinese natural-language logic problem. Both formalizations operate over the same atom symbols and translations (fixedFL). Your job is to decide whether the single specific difference flagged by Z3 is a defensible formalization choice rather than an error. Treat the original NL problem as authoritative; use the checklist desc/reason context only as secondary guidance for intended interpretation and acceptable variants. Return strict JSON only: {"softpass": true|false, "reason": "brief reason, <=30 chars"}. Do NOT analyze multiple premises at once. Do NOT include any text outside the JSON object.

User Prompt — PE Soft Review (extra premises)

The candidate added one premise that is NOT entailed by the conjunction of the gold premises. Decide if this extra premise is a defensible reading of the NL source.
Source NL problem (background): <ORIGINAL_BACKGROUND>
Question: <ORIGINAL_QUESTION>
Checklist desc/reason context (secondary guidance; original NL is authoritative): <RUBRIC_CONTEXT_JSON>
Atom translations (shared by gold and candidate): <TRANSLATION_DICT>
Gold formalization premises: <GOLD_PREMISES_LIST>
Candidate added an extra premise NOT entailed by gold: <EXTRA_PREMISE_RAW>
Decide softpass iff one of the following is clearly true: - (a) The extra premise restates a fact directly given in the NL background. - (b) The extra premise is a natural-language pragmatic implicature obviously intended in this context (e.g. "either A or B" in an exclusive context implies mutual exclusion). - (c) The extra premise is a fact that gold simply omitted from formalization but is unambiguous from the source. - (d) The checklist desc/reason explicitly allows this stronger or alternative reading, and that allowance does not contradict the NL source.
IMPORTANT: adding any constraint not entailed by gold is by default an unsupported assumption. Softpass requires a clear NL-source or checklist-desc justification (a, b, c, or d). 'It does not affect the question's answer' is NOT sufficient on its own; an unjustified extra premise still misrepresents the source even if the answer happens to coincide. If the checklist context conflicts with the NL source, prefer the NL source. In particular: - If the stronger form changes the satisfiability of the premise set (e.g. the gold premises are satisfiable but adding the extra premise makes them unsatisfiable, or vice versa), softpass=false. - If the stronger form adds new consequences that affect the question's answer, softpass=false. When in doubt, softpass=false.
Otherwise (the extra premise introduces an unstated assumption, narrows the model space without source justification, or is the candidate's own invention) -> softpass=false.
Output strict JSON: {"softpass": true|false, "reason": "brief reason, non-empty"}

User Prompt — PE Soft Review (missing premises)

The candidate's premise set does NOT entail one specific gold premise. Decide if the candidate expresses the same logical content via an equivalent or acceptable alternative form.
Source NL problem (background): <ORIGINAL_BACKGROUND>
Question: <ORIGINAL_QUESTION>
Checklist desc/reason context (secondary guidance; original NL is authoritative): <RUBRIC_CONTEXT_JSON>
Atom translations (shared by gold and candidate): <TRANSLATION_DICT>
Gold premise that candidate does NOT entail: <MISSING_PREMISE_RAW>
Candidate's full premise set: <CAND_PREMISES_LIST>
Decide softpass iff one of the following is clearly true: - (a) The candidate captures the same content via an equivalent rewriting that Z3 missed (e.g. a different quantifier formulation, a contrapositive, distributing implication over disjunction). - (b) The gold premise is split across multiple candidate premises that together imply it (only count this when the candidate clearly intends the same content). - (c) The gold premise is logically redundant for the question being formalized — its role is fully covered by the rest of the candidate set. - (d) The gold premise expresses the same content as a candidate premise written in a different abstraction layer (e.g. a rule expressed as ground facts that cover all relevant instances in this problem). - (e) The checklist desc/reason explicitly allows this weaker, alternative, or differently-scoped modeling choice, and that allowance does not contradict the NL source.
IMPORTANT: omitting a gold premise is by default a formalization gap. Softpass requires that the candidate still captures the same logical content through one of (a)-(e). Logical weakening is NOT automatically softpass; if the checklist context conflicts with the NL source, prefer the NL source. In particular: - If the weaker form changes the satisfiability of the premise set (e.g. gold premises are unsatisfiable/contradictory but the candidate's weaker premises become satisfiable, or vice versa), softpass=false. - If the weaker form loses consequences that affect the question's answer, softpass=false. When in doubt, softpass=false.
Otherwise (the candidate genuinely lacks the content of the missing gold premise, or only partially captures it, or expresses it incorrectly) -> softpass=false.
Output strict JSON: {"softpass": true|false, "reason": "brief reason, non-empty"}

Figure 26: Premise-equivalence soft review prompt for fixed-FL rubric mode. The system prompt defines the defensible-formalization standard. Two user prompt templates handle extra premises (candidate adds a premise not entailed by gold) and missing premises (candidate does not entail a gold premise) respectively.