Learning Verifiable Reasoning Programs through Cohort Consistency

Anonymous ACL submission

Abstract

In high-stakes settings, a correct answer from a large language model (LLM) is not enough: the decision procedure must also be auditable. Free-form chain-of-thought is poorly suited to this: it is a natural-language rationale rather than an explicit rule, and may not faithfully reflect the computation that produced the answer. We instead have the model emit a short executable program whose execution defines the answer, making the decision logic explicit and checkable. The central challenge is not writing runnable code but producing logic that is complete and reusable rather than an instance-specific shortcut. Our hypothesis is that a genuine reasoning rule survives factual substitution while a shortcut does not; hence consistency across a cohort of factually varied questions is a verifiable signal of reuse. We therefore train a single program to run unchanged across each cohort, using consistency as the main reinforcement-learning signal. Across five in-domain and three out-of-domain benchmarks at two model scales, our method delivers 10–20 absolute-point gains over strong vanilla, SFT, and per-instance RL baselines. Together, these results show that cohort consistency is an effective, checkable signal for learning complete and reusable reasoning.

1 Introduction[1]

In a high-stakes setting, a correct answer from a large language model (LLM) is not enough. A reviewer should be able to inspect the decision procedure the system is committing to: which facts were checked, which conditions mattered, and how those checks determined the conclusion. Free-form chain-of-thought (CoT) is poorly suited to this requirement. Although CoT prompting and related decoding strategies can improve accuracy (Wei et al., 2022; Wang et al., 2023), the resulting rationale is still natural language rather than an explicit rule, and prior work shows that such rationales can be unfaithful to the computation that produced the answer or can rely on superficial shortcuts while still sounding plausible (Yu et al., 2024; Zhou et al., 2024; Li et al., 2024b; McCoy et al., 2019; Geirhos et al., 2020). We therefore want the model to output an explicit artifact whose execution determines the answer, whose decision logic can be audited after the fact, and whose reliability can be *checked* rather than assumed.

What should that artifact be? We argue that short executable programs are a natural sweet spot. A program makes its predicates, branches, and execution semantics explicit, so the final answer is defined by running the code rather than by interpreting prose. At the same time, short programs are much lighter than full theorem-proving or planning formalisms such as Lean or PDDL (de Moura and Ullrich, 2021; McDermott et al., 1998), and more explicit than textual reasoning traces or code-assisted reasoning pipelines (Gao et al., 2022; Chen et al., 2023; Yao et al., 2023). Unlike prior program-aided reasoning, where code mainly helps solve a single instance, here the program itself is the object we want to inspect, evaluate, and improve. In the logic-heavy tasks we study, this interface is already expressive enough: questions typically reduce to a small number of factual checks with simple control flow. Given a trusted fact source, the program then serves as the auditable decision procedure that the system commits to.

The central difficulty, however, is not getting models to emit runnable code. It is getting them to emit the *right* code. Our evidence suggests that the failure is usually conceptual rather than syntactic: in an upper-bound study, pass@128 with minor manual correction yields about 94% upper-bound performance across domains, yet actual accuracy with self-consistency remains low (20-30%). The gap is logical completeness: models frequently produce programs that execute successfully yet

[1] Code and data will be released with the camera-ready version.

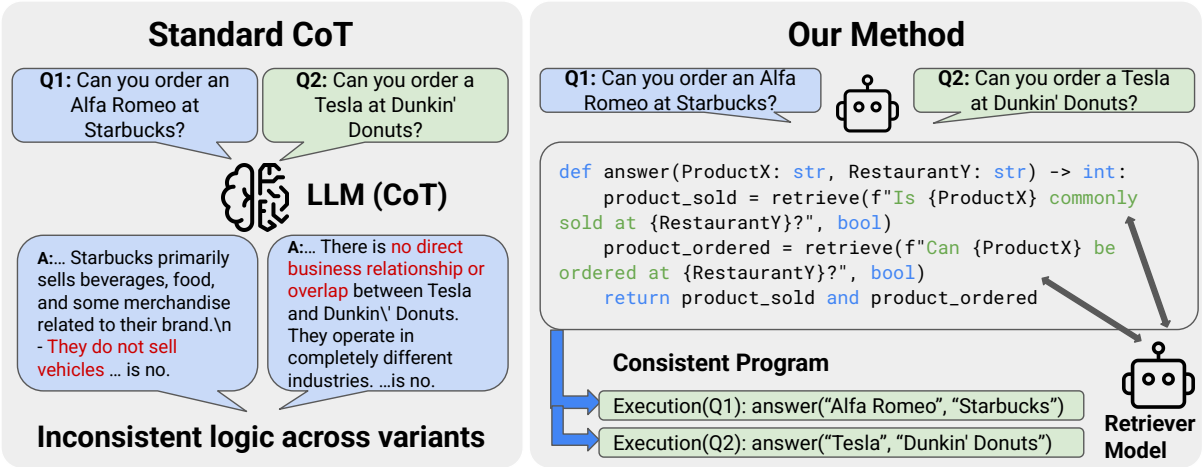

Figure 1: Why free-form CoT is poorly suited to auditing (see Appendix A.3). Standard CoT can answer closely related questions with different partial rationales. Our method instead synthesizes one executable program whose `retrieve` calls and control flow are reused unchanged across the cohort, so only the retrieved facts vary.

omit a relevant condition, compile only part of the intended reasoning, or exploit a shortcut that happens to work on one instance. Figure 1 illustrates this failure mode on StrategyQA (Geva et al., 2021): two related questions can both be answered correctly, yet via different partial rules. Single-question accuracy does not reveal the problem, because an incomplete procedure can still be correct when the missing factor happens not to matter (McCoy et al., 2019; Geirhos et al., 2020). The problem we address is therefore how to train LLMs to produce programs that are not only executable, but complete enough to be reused.

Once reasoning is represented as a program, consistency across related questions becomes a direct and *verifiable* way to solve this problem. A real decision rule should survive small factual changes; an instance-specific shortcut should not. Programs make this test especially clean: the same code can be executed unchanged across a set of questions that share the same reasoning structure while differing in entities or facts. If one program continues to work across the cohort, that is evidence that it captures the underlying rule rather than an accident of a single example. Consistency is therefore not an auxiliary metric added on top of programmatic reasoning; it is the most direct training signal for learning reasoning that is complete and reusable, and unlike faithfulness it is checkable by execution.

Motivated by this intuition, we start from a masked abstraction of a question, and we generate a cohort of factual variants that share the same reasoning pattern. During training, the policy must emit a *single* executable program that runs unchanged across the entire cohort. Programs may use only atomic `retrieve` calls plus simple control flow, so the reasoning steps stay fixed while only the retrieved facts vary. We add a rejection filter to block shortcut queries, and we use judge critique to reward decompositions that cover the relevant factors and align with an improved program structure. Reinforcement learning then optimizes this combined signal, encouraging programs that are both executable and cohort-consistent.

Across five in-domain benchmarks—ARC-Easy, ARC-Challenge (Clark et al., 2018), CommonsenseQA (Talmor et al., 2019), StrategyQA (Geva et al., 2021), and HotpotQA (Yang et al., 2018)—our method substantially improves cohort-based accuracy over vanilla and SFT baselines, with the clearest gains on ARC-Easy, ARC-Challenge, CSQA, and StrategyQA. Relative to an RL baseline that rewards each question independently, consistency-driven training is superior in most settings. On 7B models, these gains are mirrored by large improvements on ARC-Easy, ARC-Challenge, CSQA, and StrategyQA. We also observe strong transfer to out-of-distribution benchmarks, such as OpenBookQA, PubMedQA, and MMLU (Mihaylov et al., 2018; Jin et al., 2019; Hendrycks et al., 2021a,b), where consistency-trained models match or exceed the strongest RL baseline and clearly outperform SFT on single instance accuracy. Ablations further show that the improvements come from training on cohorts rather than from RL alone. Taken together, these results support a simple conclusion: short programs are an auditable, executable interface, and consis-

tency across related questions is an effective way to learn programs that are more complete, stable, and reusable.

2 Related Work

Reasoning Consistency LLMs often exhibit inconsistent reasoning when faced with paraphrased inputs. For example, prompts with similar surface familiarity but different underlying complexity yield divergent performance (Li et al., 2024a), and models may exploit spurious semantic cues rather than following intended chains of reasoning (Li et al., 2024b, 2025). To mitigate inconsistency, researchers have added training regularizers or auxiliary losses for paraphrase-invariance (Elazar et al., 2021; Zhou et al., 2022), leveraged knowledge graphs to generate paired questions for fine-tuning (Rajan et al., 2024), and applied self-consistency decoding to vote out illogical paths (Wang et al., 2023; Wei et al., 2022). Recent directions construct paraphrase/symmetry cohorts either at inference time (Chen et al., 2024) or for fine-tuning (Yao et al., 2025a) to enforce that semantically equivalent inputs yield consistent outputs (Raj et al., 2025; Prasad et al., 2025). Closest to us, Yao et al. (2025b) and Lee et al. (2025) enforce consistency over natural-language queries or rationales; we instead reuse a *single* executable program across a factually varied cohort, making consistency execution-verifiable.

Programmatic Abstractions for Reasoning Programmatic abstractions introduce a formal structure that can be executed and verified, improving transparency and reliability (Chen et al., 2024). Prior work translates questions into executable programs for verifiable stepwise reasoning (Gao et al., 2022; Chen et al., 2023), and ReAct interleaves reasoning with tool use to ground intermediate steps (Yao et al., 2023). (Zhou et al., 2024, 2025) pursue conceptual/symbolic formulations, while (Hong et al., 2024) requires an Abstraction-of-Thought plan before refining a concrete solution. More recent work grounds reasoning in execution determinism for verifiable step-level supervision (Jung et al., 2025). Our approach similarly emits compact executable programs but differs by executing a *single* program unchanged across similar questions.

Reinforcement Learning for Enhanced Reasoning. Reinforcement learning objectives can substantially boost an LLM's ability to solve multi-step problems by optimizing the reasoning path (Shen et al., 2025b; Xu et al., 2025b). Verifiable intermediate rewards catch and correct logical mistakes, leading to more stable reasoning (Xu et al., 2025a). Composite reward functions that blend answer accuracy, factuality, and faithfulness yield more dependable outputs (Wang et al., 2024), and potential-based shaping adds domain priors as soft constraints, speeding up training while keeping policies optimal (Nguyen et al., 2020). External verifiers and frozen judges have improved reliability by critiquing or selecting among candidate solutions (Cobbe et al., 2021; Zhang et al., 2025; Yang et al., 2025), and recent RL systems explicitly train models to internalize search-like, stepwise procedures (Shen et al., 2025a). Together, these advances in RL-driven reasoning path optimization, verifiable reward design, and structured shaping inspire our cohort-level consistency framework.

3 Method

As shown in Figure 2, we first convert each question into a masked abstraction and instantiate a cohort of similar questions that share the same reasoning path (see Sec. 3.1 and Sec. 3.2). The policy is trained to emit one executable program that runs unchanged across the entire cohort; the program's only external operation is an atomic `retrieve(q, type)` call, which forces the reasoning path into code and keeps the set of `retrieve` calls invariant across variants. To prevent degenerate shortcuts, the retriever is fronted by a rejection prompt—ill-formed or non-atomic queries yield "idk" and incur a penalty. We optimize with GRPO on a composite signal mixing execution reward and critique reward using a simple three-role setup (policy, retriever, judge) (See Sec. 3.3). At test time, each synthesized program is executed over its cohort and scored under lenient ($\geq 4/6$) and strict ($\geq 5/6$) criteria (Sec. 3.4). Appendix B gives the formal theory and proofs.

3.1 Program Generation

We encode each reasoning path as a small Python function `def answer(param1: Type1, ..., paramK: TypeK)->int` whose body may use only atomic `retrieve(question: str, type)` calls and control flow. The model is given (i) a masked abstraction preserving logical structure, (ii) parameter names, (iii) the answer options, and (iv) the exact function header. The fixed header specifies

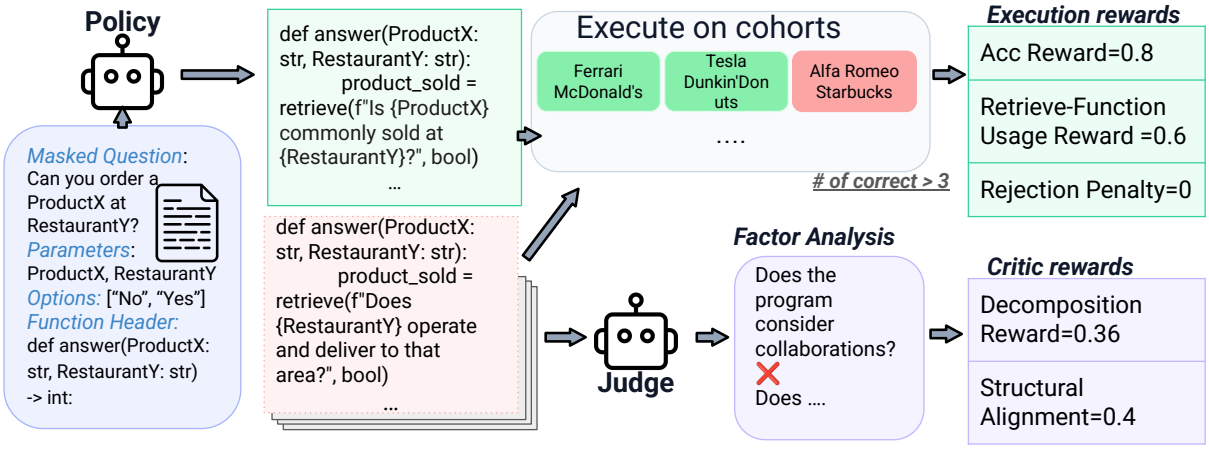

Figure 2: Overview of the training. For each masked abstraction, the policy emits a single program p that uses only atomic `retrieve(q, type)` and is executed *unchanged* across a cohort of questions. Execution over the cohort yields the Accuracy Reward, the Retrieve-Function Usage Reward and Rejection Penalty. A judge supplies critique signals: the Factor-Complete Decomposition Reward and the Structural Alignment Reward. These combine into the composite RL objective.

parameter names and types as well as the return type, which (i) tells the model the concrete type of each input, guiding clearer program generation; and (ii) ensures unambiguous execution at evaluation time. Four few-shot exemplars—boolean checks, numeric comparisons, list loops, and dependent lookups—specify the output format across cohort abstractions (Appendix A.1.5). At generation time the policy fills in the function; at execution time each `retrieve` is issued to the retriever.

3.2 Data Generation and Preprocessing

To foster generalizable reasoning and provide a strong foundation for our models, we employ a high-quality data preparation pipeline.

Similar Questions Generation Central to our approach is the construction of cohorts of similar questions that share same reasoning paths but differ in factual content. This process begins with a corpus of 5,000 original questions randomly sampled from each domain's training split (1,000 per domain). As Figure 1 shows, for each original question, we generate an abstraction that preserves its core reasoning structure while parameterizing entities, allowing for the substitution of different facts (see Appendix A.1.2). Using LLaMA-3.3-70B-Instruct (Team, 2024a), we then instantiate 5 similar questions for each abstraction (see Appendix A.1.3). To ensure the fidelity of our training data, answers to all generated similar questions are cross-validated using three state-of-the-art models: LLaMA-3.3-70B-Instruct, DeepSeek-R1-Distill-Llama-70B(DeepSeek-AI, 2025), and Qwen-2.5-72B-Instruct(Yang et al., 2024; Team, 2024b). This process is supplemented by human verification, as detailed in 4.5.

Program Corpus for Supervised Fine-Tuning (SFT). We construct two corpora corresponding to the SFT variants. For $\text{SFT}_{\text{HQ-500}}$, we synthesize 500 programs (100 per domain) with LLaMA-3.3-70B-Instruct and rigorously verify each to achieve **0% rejection** and **100% execution accuracy** on its question. For $\text{SFT}_{\text{DM-5k}}$, we reuse the 5,000 RL-training instances and generate programs with the same 70B model and prompts; to keep computational cost tractable at this scale, we retain basic validity checks but do not perform the exhaustive curation needed to guarantee 0% rejection.

3.3 Consistent Program Learning

We optimize the policy with Group Relative Policy Optimization (GRPO) on cohorts of six (1 original + 5 similar). For each abstraction, the policy emits a single executable program p that is executed *unchanged* across all variants. A retriever answers atomic `retrieve` calls, and a rejection filter blocks multi-hop or invalid queries. The policy then receives a composite reward R. We group the rewards into two families: (i) *execution-based* rewards (Accuracy Reward, the Retrieve-Function Usage Reward and Rejection Penalty), which are verifiable from program execution and the retriever/rejection outcomes; and (ii) *critique-based* rewards (the Factor-Complete Decomposition Reward and the Structural Alignment Reward), which are produced by a frozen judge. Together, these re-

wards guide the policy toward accurate, disciplined decomposition and convergence to factor-complete reasoning procedures .

Model Architecture We evaluate both 3B and 7B model sizes throughout. Our framework employs three language models: **(a) Policy Model:** Qwen-2.5-Coder-Instruct (3B/7B) (Yang et al., 2024; Team, 2024b; Hui et al., 2024) generates structured, executable programs for abstracted questions, specifying the reasoning path. **(b) Retriever Model:** Qwen-2.5-Instruct (3B/7B) serves as our retriever model; it executes the simple retrieval calls generated within the programs by the policy model. **(c) Judge Model:** the same Qwen-2.5-Instruct (3B/7B) checkpoint as the retriever, used as an evaluator to score whether sub-questions cover key factors and to propose a concise improved program p^+ for logic-level alignment checking. This architecture is identical across scales and ensures that the policy must formulate its reasoning strategy without direct access to factual information during program generation.

Rejection Prompts in Retrieval To encourage the policy model to learn robust, generalizable reasoning and prevent the policy model from circumventing the intended reasoning process by issuing trivial or multi-step queries—behavior akin to the "deceptive shortcuts" observed in prior work (Li et al., 2024b)—we equip the retriever model with a rejection-prompt filter. With a few-shot prompt (see Appendix A.1.1), the retriever only accepts straightforward, single-step factual questions (e.g., Is {ProductX} sold at {RestaurantY}?) and replies with "idk" to any non-atomic queries.

Interactive Training Pipeline The GRPO training loop proceeds interactively. First, the policy model proposes a program in response to an abstracted question. Second, this program is executed on both the original question and its five similar variants. Any information retrieval calls embedded in the program are handled by the retriever model. Finally, based on the execution outcomes across the entire question family, a scalar reward is computed. This reward considers accuracy, retrieval efficiency, and the rejection ratios.

Reward Components Our composite reward $R = R_{\text{acc}} + R_{\text{ret}} + R_{\text{rej}} + R_{\text{fc}} + R_{\text{sa}}$ combines *execution-based* signals $\left(R_{\text{acc}}, R_{\text{ret}}, R_{\text{rej}}\right)$ with *critique-based* signals $\left(R_{\text{fc}}, R_{\text{sa}}\right)$ from a frozen judge model. We use *reward scale separation*: every non-accuracy component is normalized to share a common magnitude cap C, while the accuracy term spans $[0, 2C]$, so accuracy is prioritized by construction. **Accuracy Reward (R_{acc}):** This reward directly optimizes cohort-level correctness. **Retrieve-Function Usage Reward (R_{ret}):** This term encourages meaningful decomposition using `retrieve` calls (and discourages trivial behavior with no retrieval). **Rejection Penalty (R_{rej}):** To deter shortcut or invalid retrieval queries (e.g., repeating the input question or near-duplicates that get rejected by the retrieval model), we penalize rejected calls at the cohort level. **Factor-Complete Decomposition Reward (R_{fc}):** Even when execution feedback is sparse (e.g., some cohort members are wrong), we want to reward decompositions that nevertheless cover the key factors needed to solve the abstraction. Given the program, the cohort, and observed failure patterns, a frozen judge assigns a coverage score indicating whether the `retrieve` sub-questions are factor-complete. **Structural Alignment Reward (R_{sa}):** To encourage reflective improvement at the level of reasoning logic (not formatting), the judge proposes an improved program p^+ given the full context. We then compare the original program p to p^+ on structural similarity. (Appendix A.4 gives the detailed design for each reward.)

RL Variants To isolate how the accuracy signal shapes learning, we consider two training-time variants that differ only in how R_{acc} is computed (all other terms unchanged; to avoid confusion, *lenient/strict* refer only to evaluation criteria): **(a) Cohort Accuracy:** Accuracy-based rewards (R_{acc}) are granted only if the generated program successfully answers at least 4 out of the 6 questions in one group. This enforces a higher standard of generalizability. **(b) Normal Accuracy:** Accuracy rewards (R_{acc}) are granted for every successful program execution on a question within the group, regardless of performance on other questions in that group. This provides a more granular learning signal.

3.4 Cohort Execution Test

We evaluate each program on a cohort of six questions (original + five variants). We report two criteria: **Strict Accuracy:** the program is correct iff it answers ≥ 5 of 6; **Lenient Accuracy:** correct iff it answers ≥ 4 of 6. In addition, we apply a lightweight rejection check at evaluation to prevent

degenerate behavior that simply re-asks the original question or violates the simple-fact constraint; any instance with a rejected `retrieve` call is counted as incorrect under both criteria. We use ≥ 5 of 6 instead of 6 of 6 for strict accuracy is to make the metric robust to occasional label noise in the generated questions. Our human audit finds 92% answer accuracy (Table 10), so even a perfectly consistent program may sometimes miss one item due to annotation error.

4 Experiments

In this section, we provide detailed experimental settings and results that highlight the effectiveness of our RL framework in training LLMs to perform reasoning through structured, executable programs. The full set of hyperparameters is listed in Appendix A.2.

Baselines. We compare our consistency-based RL models against two configurations: **(a) Vanilla Model:** the off-the-shelf Qwen-2.5-Coder-Instruct checkpoint, used without any additional supervised fine-tuning or RL. **(b) Supervised Fine-Tuning (SFT):** we consider two SFT variants on the same backbone. $\text{SFT}_{\text{HQ-500}}$ ("High Quality SFT") fine-tunes on a curated set of 500 exemplar programs (100 per domain) until held-out loss stabilizes (typically $\sim$1.1 epochs; see Sec. 3.2). $\text{SFT}_{\text{DM-5k}}$ ("Data-Matched SFT") fine-tunes on the same 5,000 instances used for RL, using programs synthesized by LLaMA-3.3-70B-Instruct with the same prompting but without exhaustive curation, isolating the effect of RL's cohort-level credit assignment from simply scaling SFT on the RL data distribution.

4.1 Experimental Setup

Datasets and Test Set We evaluate on five public benchmarks: ARC-Easy, ARC-Challenge, CSQA, StrategyQA, and HotpotQA. Our test set comprises random 2,500 questions in total (500 per dataset, randomly sampled from dev/test split), each paired with five similar questions generated by our abstraction pipeline.

Evaluation Protocol For each of the 2,500 test questions, we sample 11 times and execute each sampled program on its six-question cohort, and aggregate predictions via self-consistency: the final answer is the majority vote over the 11 runs. We then compute strict and lenient accuracy as defined in Sec. 3.4. We also report the standard single-question accuracy in Appendix A.11.

4.2 Main Results

On 7B, $\text{RL}_{\text{Consistency}}$ (Exec+Crit) improves over the strongest SFT baseline ($\text{SFT}_{\text{DM-5k}}$) by +33.2 on ARC-Easy (74.8 vs. 41.6), +37.4 on CSQA (73.4 vs. 36.0), and +28.2 on StrategyQA (45.8 vs. 17.6) under the lenient metric; Under the strict metric, gains remain large: +36.8 (ARC-Easy: 68.0 vs. 31.2), +35.2 (CSQA: 62.4 vs. 27.2), +22.4 (StrategyQA: 31.0 vs. 8.6), and +36.2 (ARC-Challenge: 56.6 vs. 20.4). HotpotQA and ARC-Challenge are exceptions: under the lenient metric, $\text{RL}_{\text{Consistency}}$ (Exec+Crit) is slightly below the strongest RL variant. Under the strict metric, HotpotQA still trails (-5.2: 54.2 vs. 59.4 with execution-only), while ARC-Challenge is effectively tied (56.6 vs. 56.2).

On 3B, $\text{RL}_{\text{Consistency}}$ (Exec+Crit) improves over $\text{SFT}_{\text{DM-5k}}$ by +20.2 (ARC-Easy: 40.8 vs. 20.6), +28.6 (ARC-Challenge: 41.8 vs. 13.2), and +15.0 (CSQA: 42.6 vs. 27.6)under the lenient metric; strict gains are +23.0 (ARC-Easy), +26.6 (ARC-Challenge), +17.4 (CSQA), and +31.0 (StrategyQA). HotpotQA shows little change at 3B (–3.2 lenient; $+0.8$ strict). Taken together (Tables 1 and 2), these gains are consistent across criteria and model sizes, indicating that enforcing a single program with critique yields more stable, consistent reasoning on complex, multi-step questions.

4.3 Sanity Check Experiments

Reject Prompts Analysis We validate the retriever's rejection prompts by measuring rejection ratios on 5k training questions and on SimpleQA (Wei et al., 2024) to check over-rejection. As shown in Table 3, the mechanism efficiently distinguishes different questions.

Upper Bound Analysis To assess the theoretical feasibility of our evaluation protocol, we estimate an upper-bound performance by running pass@128 on a random subset of 50 questions per domain and then manually correcting any incorrect programs. As shown in Table 3, after this minor intervention the model reaches near-perfect accuracy. This demonstrates that the evaluation itself is not an insurmountable "mission impossible"; rather, the gap underscores that existing LLMs still require significant advances in consistency.

4.4 Ablation Studies

Training on Original Questions Only. To further evaluate the effectiveness of the similar question during training, we conducted an ablation

Model	Methods	ARC-C	ARC-E	CSQA	StrategyQA	HotpotQA
Qwen2.5-Coder-7B-Instruct	Vanilla	19.0 ± 3.1	30.0 ± 4.0	29.8 ± 4.0	12.6 ± 2.9	45.0 ± 4.3
	$\text{SFT}_{\text{HQ-500}}$	19.8 ± 3.5	33.4 ± 4.1	32.0 ± 4.1	12.0 ± 2.9	46.8 ± 4.4
	$\text{SFT}_{\text{DM-5k}}$	28.8 ± 4.0	41.6 ± 4.3	36.0 ± 4.2	17.6 ± 3.3	77.6 ± 3.7
	$\text{RL}_{\text{Normal}}$ (Acc)	30.2 ± 4.0	43.4 ± 4.3	36.0 ± 4.2	15.8 ± 3.2	73.2 ± 3.8
	$\text{RL}_{\text{Consistency}}$ (Exec)	51.0 ± 4.4	60.8 ± 4.3	59.6 ± 4.3	38.2 ± 4.2	$\mathbf{81.4 \pm 3.4}$
	$\text{RL}_{\text{Normal}}$ (Exec+Crit)	$\mathbf{66.2 \pm 4.1}$	72.6 ± 3.9	68.0 ± 4.1	43.6 ± 4.3	81.0 ± 3.4
	$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	65.8 ± 4.1	$\mathbf{74.8 \pm 3.8}$	$\mathbf{73.4 \pm 3.9}$	$\mathbf{45.8 \pm 4.4}$	79.4 ± 3.5
Qwen2.5-Coder-3B-Instruct	Vanilla	12.2 ± 2.9	14.4 ± 3.1	12.0 ± 2.9	5.4 ± 2.0	17.0 ± 3.3
	$\text{SFT}_{\text{DM-5k}}$	13.2 ± 3.0	20.6 ± 3.5	27.6 ± 3.9	3.6 ± 1.7	$\mathbf{18.2 \pm 3.4}$
	$\text{RL}_{\text{Consistency}}$ (Exec)	31.6 ± 4.1	30.4 ± 4.0	38.0 ± 4.2	1.8 ± 1.2	0.4 ± 0.7
	$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	$\mathbf{41.8 \pm 4.3}$	$\mathbf{40.8 \pm 4.3}$	$\mathbf{42.6 \pm 4.3}$	$\mathbf{44.2 \pm 4.3}$	15.0 ± 3.1

Table 1: Lenient Accuracy (%) across datasets. Vanilla: off-the-shelf Qwen-2.5-Coder-Instruct. $\text{SFT}_{\text{HQ-500}}$/$\text{SFT}_{\text{DM-5k}}$: curated 500 exemplars vs. data-matched 5k set. $\text{RL}_{\text{Normal}}$ (Acc): RL with accuracy-only reward (R_{acc}; baseline). $\text{RL}_{\text{Consistency}}$ (Exec): RL with execution-based rewards only ($R_{\text{acc}}, R_{\text{ret}}, R_{\text{rej}}$). $\text{RL}_{\text{Normal}}$/$\text{RL}_{\text{Consistency}}$(Exec+Crit): RL using both execution- and critique-based rewards ($R_{\text{fc}}, R_{\text{sa}}$), with per-instance vs. cohort-gated accuracy. Here $\pm$ denotes the half-width of the 95% Wilson confidence interval. **Bold** = best, underline = second best.

Model	Methods	ARC-C	ARC-E	CSQA	StrategyQA	HotpotQA
Qwen2.5-Coder-7B-Instruct	Vanilla	13.6 ± 2.9	22.6 ± 3.7	20.6 ± 3.5	6.8 ± 2.2	27.2 ± 3.9
	$\text{SFT}_{\text{HQ-500}}$	14.4 ± 3.1	24.2 ± 3.7	25.0 ± 3.8	6.2 ± 2.1	27.8 ± 3.9
	$\text{SFT}_{\text{DM-5k}}$	20.4 ± 3.5	31.2 ± 4.1	27.2 ± 3.9	8.6 ± 2.5	53.2 ± 4.3
	$\text{RL}_{\text{Normal}}$ (Acc)	22.4 ± 3.7	35.4 ± 4.2	27.6 ± 3.9	8.4 ± 2.4	49.8 ± 4.4
	$\text{RL}_{\text{Consistency}}$ (Exec)	40.6 ± 4.3	51.2 ± 4.4	48.4 ± 4.4	22.2 ± 3.7	$\mathbf{59.4 \pm 4.3}$
	$\text{RL}_{\text{Normal}}$ (Exec+Crit)	56.2 ± 4.3	64.4 ± 4.2	59.8 ± 4.3	27.2 ± 3.9	56.2 ± 4.3
	$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	$\mathbf{56.6 \pm 4.3}$	$\mathbf{68.0 \pm 4.1}$	$\mathbf{62.4 \pm 4.2}$	$\mathbf{31.0 \pm 4.0}$	54.2 ± 4.4
Qwen2.5-Coder-3B-Instruct	Vanilla	7.6 ± 2.3	9.8 ± 2.6	8.2 ± 2.4	1.2 ± 1.0	6.0 ± 2.1
	$\text{SFT}_{\text{DM-5k}}$	8.8 ± 2.5	14.0 ± 3.0	18.8 ± 3.4	1.6 ± 1.2	6.8 ± 2.2
	$\text{RL}_{\text{Consistency}}$ (Exec)	26.4 ± 3.9	26.6 ± 3.9	32.4 ± 4.1	1.4 ± 1.1	0.2 ± 0.3
	$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	$\mathbf{35.4 \pm 4.2}$	$\mathbf{37.0 \pm 4.2}$	$\mathbf{36.2 \pm 4.2}$	$\mathbf{32.6 \pm 4.1}$	$\mathbf{7.6 \pm 2.3}$

Table 2: Strict Accuracy (%) across datasets. Naming and notes follow Table 1. **Bold** = best, underline = second best.

study where RL training was performed solely on the original questions without considering the similar variants. Table 4 and Table 5 compare the performance of this approach with our Consistency RL variant. The results demonstrate consistent performance degradation when training solely on original questions, particularly for complex reasoning tasks like ARC-Challenge and CSQA. This confirms that the similar questions play a crucial role in compelling the model to learn generalizable reasoning strategies.

High-Quality Retriever & Judge To further probe how retriever/judge strength affects training, we run a controlled study where all models are trained at 3B. Concretely: 3B-3B trains with a 3B retriever+judge and evaluates with a 3B retriever; 3B-7B trains with a 3B retriever+judge but evaluates with a 7B retriever; 7B-7B trains with a 7B retriever+judge and evaluates with a 7B retriever. As shown in Table 6 and Table 7, 7B-7B yields the strongest scores on most datasets, while 3B-7B is consistently second-best and improves over 3B-3B. An exception is StrategyQA, where lighter judges (3B-3B / 3B-7B) edge out 7B-7B. Overall, stronger retrieval/judging generally boosts reasoning.

Out Of Domain Data Accuracy To assess generalization beyond our training domains, we evaluated on three out of domain benchmarks—OpenBookQA, PubMedQA, and MMLU (Table 8). Because constructing reliable similar-question cohorts is nontrivial for these datasets, we report single-question accuracy only, using the same self-consistency strategy as in our in-domain evaluation. $\text{RL}_{\text{Consistency}}$ attains the best accuracy on PubMedQA and MMLU and is on par with $\text{RL}_{\text{Normal}}$ on OpenBookQA; both RL variants substantially outperform SFT and Vanilla across all three datasets. These results indicate that our method not only stabilizes in-domain reasoning but also transfers effectively to new tasks.

Reward Component Analysis. We ablate individual reward components in our composite objective $R = R_{\text{acc}} + R_{\text{ret}} + R_{\text{rej}} + R_{\text{fc}} + R_{\text{sa}}$ to understand what each term contributes to learning. The execution-based rewards play a critical role in preventing degenerate behaviors: removing the retrieve-usage reward R_{ret} causes the policy to collapse toward trivial behavior, where it tends to issue a single retrieve call that is effectively a paraphrase of the input question rather than a meaningful decomposition into atomic sub-questions. Similarly, removing the rejection penalty R_{rej} eliminates the deterrent against shortcut queries, and the policy often attempts to repeat the input question directly as a retrieve call. Both outcomes undermine our learning. In contrast, the judge-based rewards (R_{fc} and R_{sa}) are designed to provide additional shaping signals when execution feedback is sparse: even when a generated program fails on some cohort members, the judge can still reward partially useful structure, so the policy can learn from imperfect but instructive generations rather than receiving only all-or-nothing accuracy feedback.

RL Hyperparameter Analysis. We also analyze the sensitivity of training to reward weighting choices. In our default reward design, the accuracy reward is intentionally prioritized: R_{acc} is weighted to be about $2\times$ the magnitude of any other single reward. This reflects our intent to optimize cohort-level accuracy as the primary objective, while using non-accuracy rewards as secondary shaping signals that steer learning toward disciplined decomposition. When we retrain with equal weights across reward components in the 3B model, we observe a 10–20 percent drop in accuracy under lenient and strict metric (see Appendix A.10 and Table 15) because the accuracy reward quickly plateaus and stops improving during training; once this happens, the optimizer shifts its focus to increasing the other reward terms, which degrades final accuracy.

Program–CoT Comparsion Programs are not a lossless transcription of free-form CoT, but the gap is limited enough that they remain a meaningful auditable interface (See Table 9). More importantly, in a manual audit of 50 annotated question–program pairs, 68.0% were judged acceptable executable formalizations of the original question. We therefore view programs as faithful enough to support audit and learning, while still imperfect enough that consistency-based training is needed to close the remaining gap.

4.5 Human Study

Similar Question Quality Assessment To verify that our automated pipeline produces high-quality question variants, we assigned five annotators to assess a random sample of 150 generated questions (30 per domain) along two criteria: whether the answer label matches the ground truth, and whether the question follows its abstraction. As shown in Table 10, our generated similar questions have high label and abstraction accuracy ($\geq 90\%$).

Comparison of Reasoning Path We also conducted a comparison of programmatic reasoning paths from our RL-trained model versus a supervised fine-tuning (SFT) baseline. Two annotators evaluated 100 instances (20 per domain), choosing which program showed superior logical coherence, clarity, and decomposition. As shown in Table 10, the Consistency RL's reasoning paths are preferred 47% of the time over the left 2 models.

Failure Case Analysis We analyze 50 shared failure questions (10 per domain) where both SFT and RL answered incorrectly, and label each with one of three mutually exclusive types: **Ambiguity/Annotation**: the question or the label is ambiguity; **Control-Flow**: the program's explicit logic is wrong (e.g., incorrect AND/OR aggregation, quantifier handling, branching/looping); **Syntax**: The model uses incorrect syntax; **Retrieve**: misuse of the `retrieve` call. Two annotators independently labeled all items. One example is shown in Appendix A.9. As shown in the Table 11, RL reduces retrieve failures, consistent with execution signals and rejection penalties discouraging non-atomic or shortcut queries. Residual RL errors shift toward Control-Flow or Syntax. This supports our claim that cohort-gated RL suppresses shortcuts and enforces more disciplined, programmatic reasoning.

5 Conclusion

We train a single executable program to run unchanged across a cohort of similar questions, using a composite RL objective that combines execution feedback with judge critique. Across five in-domain and three out-of-domain benchmarks, our method outperforms vanilla, SFT, and per-instance RL baselines under both lenient and strict criteria, with the clearest gains on multi-step tasks and reasoning paths that human annotators prefer, which indicates that consistency across related questions It is an effective signal for reusable reasoning.

Limitations

Our study focuses on Qwen2.5-Coder models at the 3B and 7B scales, and on a set of English factual and commonsense reasoning benchmarks; results may differ for substantially larger models, other model families, or other languages and domains. Our method also relies on a retriever to resolve atomic factual queries, so its end-to-end accuracy depends in part on the quality of the underlying retrieval model, and constructing cohorts of similar questions requires an automated abstraction pipeline that may need adaptation for new tasks. We leave a broader exploration of these settings to future work.

References

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Wenqing Chen, Weicheng Wang, Zhixuan Chu, Kui Ren, Zibin Zheng, and Zhichao Lu. 2024. Self-paraconsistency: Improving reasoning tasks at low cost for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14162–14167, Bangkok, Thailand. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 625–635. Springer.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. 2021. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021a. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021b. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Ruixin Hong, Hongming Zhang, Xiaoman Pan, Dong Yu, and Changshui Zhang. 2024. Abstraction-of-thought makes language models better reasoners. *Preprint*, arXiv:2406.12442.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577, Hong Kong, China. Association for Computational Linguistics.

Dongwon Jung, Wenxuan Zhou, and Muhao Chen. 2025. Code execution as grounded supervision for llm reasoning. *Preprint*, arXiv:2506.10343.

Jaehyeok Lee, Keisuke Sakaguchi, and JinYeong Bak. 2025. Self-training meets consistency: Improving llms' reasoning with consistency-driven rationale evaluation. *Preprint*, arXiv:2411.06387.

Bangzheng Li, Ben Zhou, Xingyu Fu, Fei Wang, Dan Roth, and Muhao Chen. 2024a. Famicom: Further demystifying prompts for language models with task-agnostic performance estimation. *Preprint*, arXiv:2406.11243.

Bangzheng Li, Ben Zhou, Fei Wang, Xingyu Fu, Dan Roth, and Muhao Chen. 2024b. Deceptive semantic shortcuts on reasoning chains: How far can models go without hallucination? *Preprint*, arXiv:2311.09702.

Zhaonan Li, Shijie Lu, Fei Wang, Jacob Dineen, Xiao Ye, Zhikun Xu, Siyi Liu, Young Min Cho, Bangzheng Li, Daniel Chang, Kenny Nguyen, Qizheng Yang, Muhao Chen, and Ben Zhou. 2025. Unbiased visual reasoning with controlled visual inputs. *Preprint*, arXiv:2512.22183.

R Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *arXiv preprint arXiv:1902.01007*.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. Pddl – the planning domain definition language. AIPS-98 Planning Competition Committee Technical Report.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *Preprint*, arXiv:1809.02789.

Hoang Nguyen, Mario Araya, and Finale Doshi-Velez. 2020. Learning to utilize shaping rewards: A new approach of potential-based reward shaping. In *NeurIPS 2020*.

Archiki Prasad, Weizhe Yuan, Richard Yuanzhe Pang, Jing Xu, Maryam Fazel-Zarandi, Mohit Bansal, Sainbayar Sukhbaatar, Jason Weston, and Jane Yu. 2025. Self-consistency preference optimization. *Preprint*, arXiv:2411.04109.

Harsh Raj, Vipul Gupta, Domenic Rosati, and Subhabrata Majumdar. 2025. Improving consistency in large language models through chain of guidance. *arXiv preprint arXiv:2502.15924*.

SaiSathiesh Rajan, Ezekiel Soremekun, and Sudipta Chattopadhyay. 2024. Knowledge-based consistency testing of large language models. In *Findings of EMNLP 2024*. To appear.

Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory Wornell, Subhro Das, David Cox, and Chuang Gan. 2025a. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *Preprint*, arXiv:2502.02508.

Ming Shen, Zhikun Xu, Xiao Ye, Jacob Dineen, and Ben Zhou. 2025b. Bow: Bottlenecked next word exploration. *Preprint*, arXiv:2506.13502.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297. ACM.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

LLaMA Team. 2024a. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Qwen Team. 2024b. Qwen2.5: A party of foundation models.

Haiyang Wang, Yuchen Pan, Xin Song, Xuechen Zhao, Minghao Hu, and Bin Zhou. 2024. F^2rl: Factuality and faithfulness reinforcement learning framework for claim-guided evidence-supported counterspeech generation. In *EMNLP 2024*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, and Aleksei Gusev. 2023. Self-consistency improves chain-of-thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. 2024. Measuring short-form factuality in large language models. *Preprint*, arXiv:2411.04368.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Yifei Xu, Tusher Chakraborty, Srinagesh Sharma, and et al. 2025a. Direct reasoning optimization: Llms can reward and refine their own reasoning for open-ended tasks. arXiv preprint arXiv:2506.13351.

Zhikun Xu, Ming Shen, Jacob Dineen, Zhaonan Li, Xiao Ye, Shijie Lu, Aswin RRV, Chitta Baral, and Ben Zhou. 2025b. Tow: Thoughts of words improve reasoning in large language models. *Preprint*, arXiv:2410.16235.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Diji Yang, Linda Zeng, Kezhen Chen, and Yi Zhang. 2025. Beyond introspection: Reinforcing thinking via externalist behavioral feedback. *Preprint*, arXiv:2501.01457.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2369–2380. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*.

Yihang Yao, Zhepeng Cen, Miao Li, William Han, Yuyou Zhang, Emerson Liu, Zuxin Liu, Chuang Gan, and Ding Zhao. 2025a. Your language model may think too rigidly: Achieving reasoning consistency with symmetry-enhanced training. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 23036–23052, Vienna, Austria. Association for Computational Linguistics.

Yihang Yao, Zhepeng Cen, Miao Li, William Han, Yuyou Zhang, Emerson Liu, Zuxin Liu, Chuang Gan, and Ding Zhao. 2025b. Your language model may think too rigidly: Achieving reasoning consistency with symmetry-enhanced training. *Preprint*, arXiv:2502.17800.

Xiaodong Yu, Ben Zhou, Hao Cheng, and Dan Roth. 2024. Reasonagain: Using extractable symbolic programs to evaluate mathematical reasoning. *Preprint*, arXiv:2410.19056.

Xiaoying Zhang, Hao Sun, Yipeng Zhang, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. 2025. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback. *Preprint*, arXiv:2506.03106.

Ben Zhou, Sarthak Jain, Yi Zhang, Qiang Ning, Shuai Wang, Yassine Benajiba, and Dan Roth. 2025. Self-supervised analogical learning using language models. *Preprint*, arXiv:2502.00996.

Ben Zhou, Hongming Zhang, Sihao Chen, Dian Yu, Hongwei Wang, Baolin Peng, Dan Roth, and Dong Yu. 2024. Conceptual and unbiased reasoning in language models. *Preprint*, arXiv:2404.00205.

Chunting Zhou, Junxian He, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Prompt consistency for zero-shot task generalization. *Preprint*, arXiv:2205.00049.

A Appendix

A.1 Prompts

A.1.1 Rejection Prompt

Prompt for *Fact-Lookup Assistant*

System: You are a fact-lookup assistant. For each user query, first decide if it's a simple, single-step fact lookup without solving it and then return a JSON object with exactly one key, "answer", wrapped in "'json ..."'. Match the type specified in parentheses (int, str, list, bool). If a query requires more than a straightforward fact check or true/false lookup—for example, multi-step reasoning or subjective judgment—reply with "idk".

User: Who finished immediately after the winner at the 1992 Olympic 100m final? (str)
Assistant: [Explanation] You must identify the winner, then determine who came second—this isn't single-step. "'json "answer": "idk" "'

User: How many planets are in the solar system? (int)
Assistant: [Explanation] Simple fact check. "'json "answer": 8 "'

User: What is the profession of Michael Jackson? (str)
Assistant: [Explanation] Single well-known profession of a public figure. "'json "answer": "singer" "'

User: Who has more than one Nobel Prize? (list)
Assistant: [Explanation] Factual list of individuals with multiple Nobel Prizes. "'json "answer": ["John Bardeen", "Frederick Sanger", "Linus Pauling", "Marie Curie"] "'

User: Is the CEO of Tesla older than the current President of France? (bool)
Assistant: [Explanation] Requires fetching and comparing two birthdates—multi-step. "'json "answer": "idk" "'

User: Is the Eiffel Tower located in Paris, France? (bool)
Assistant: [Explanation] Single-step landmark location. "'json "answer": true "'

User: Did England win any Olympic gold medals in 1800? (bool)
Assistant: [Explanation] Must check when the modern Olympics began and then medal records—multi-step. "'json "answer": "idk" "'

User: What is the population of the largest country entirely south of the equator? (int)
Assistant: [Explanation] Identify the country then lookup its population—multi-step. "'json "answer": "idk" "'

User: List the U.S. states admitted to the Union between the first and the last of the original 13 colonies. (list)
Assistant: [Explanation] Order states by admission date and filter—multi-step. "'json "answer": "idk" "'

User: Can food be cooked in the cosmic microwave background? (bool)
Assistant: [Explanation] Must compare CMB temperature (2.7 K) to cooking physics—multi-step. "'json "answer": "idk" "'

User: Are Waris Hussein and Mathieu Kassovitz both actors? (bool)
Assistant: [Explanation] Fetch each person's profession and compare—multi-step. "'json "answer": "idk" "'

User: {prompt} ({return_type})
Assistant:

A.1.2 Abstraction Generation Prompt

Prompt for *Masked Question Generation*

System: Generate a masked question and corresponding parameters for the given question. These generations will be used to create similar questions that share the same reasoning path and options. The similar questions can only replace the values in the parameters. Therefore, be careful with the parameters. Do not include options or answers in the parameters. Present the output in JSON format.

User: The Sugarcubes and SCH, are related to music? Options: A. No B. Yes. Answer: B
Assistant:

```
{
  "masked_question": "Are Entity1 and Entity2 both related to DomainX?",
  "parameters": {
    "Entity1": "The Sugarcubes",
    "Entity2": "SCH",
    "DomainX": "music"
  }
}
```

User: Are Ken Kwapis and Jan Švankmajer both filmmakers? Options: A. No B. Yes. Answer: B
Assistant:

```
{
  "masked_question": "Are Person1 and Person2 both ProfessionX?",
  "parameters": {
    "Person1": "Ken Kwapis",
    "Person2": "Jan Švankmajer",
    "ProfessionX": "filmmakers"
  }
}
```

User: Which of these is one similarity of the elements in the first group of the periodic table? Options: A. The elements in the group are very unreactive metals. B. The elements in the group react with oxygen in similar ways. Answer: B
Assistant:

```
{
  "masked_question": "Which of these is one similarity of the elements in the Group1 of the periodic table?",
  "parameters": {
    "Group1": "first group"
  }
}
```

User: Which celestial object listed below has the greatest density? Options: A. a planet B. a neutron star. Answer: B
Assistant:

```
{
  "masked_question": "Which celestial object listed below has the Superlative Property?",
  "parameters": {
    "Superlative": "greatest",
    "Property": "density"
  }
}
```

User: On what would you serve an egg to someone? Options: A. plate B. chicken coop. Answer: A
Assistant:

```
{
  "masked_question": "On what would you serve a FoodItem to someone?",
  "parameters": {
    "FoodItem": "egg"
  }
}
```

User: Fighting inflation means being aware of many things, such as what with loans? Options: A. rising interest rates B. loss of money. Answer: A
Assistant:

```
{
  "masked_question": "Fighting EconomicIssue means being aware of many things, such as what with FinancialAspect?",
  "parameters": {
    "EconomicIssue": "inflation",
    "FinancialAspect": "loans"
  }
}
```

User: Question: {question} Options: {options}. Answer: {answer}
Assistant:

A.1.3 Similar Question Prompt

Prompt for *Similar Question Generation*

System: Generate five new questions that follow the same underlying reasoning process based on the masked questions, parameters, and the original question. Ensure that the choices and masked question remain the same as the original. You may only change the values within the `parameters` dictionary. Present the output in JSON format.

User: The Sugarcubes and SCH, are related to music? Options: A. No B. Yes. Answer: B. Masked Question: Are Entity1 and Entity2 both related to DomainX? Parameters: 'Entity1': 'The Sugarcubes', 'Entity2': 'SCH', 'DomainX': 'music'.
Assistant:

```
{
  "similar_questions": [
    {
      "question": "Are The Beatles and ABBA both related to music?",
      "answer": "B",
      "parameters": {
        "Entity1": "The Beatles",
        "Entity2": "ABBA",
        "DomainX": "music"
      }
    },
    {
      "question": "Are Marvel and DC both related to comics?",
      "answer": "B",
      "parameters": {
        "Entity1": "Marvel",
        "Entity2": "DC",
        "DomainX": "comics"
      }
    },
    {
      "question": "Are NASA and SpaceX both related to space exploration?",
      "answer": "B",
      "parameters": {
        "Entity1": "NASA",
        "Entity2": "SpaceX",
        "DomainX": "space exploration"
      }
    },
    {
      "question": "Are Pepsi and Coca-Cola both related to beverages?",
      "answer": "B",
      "parameters": {
        "Entity1": "Pepsi",
        "Entity2": "Coca-Cola",
        "DomainX": "beverages"
      }
    },
    {
      "question": "Are Microsoft and Apple both related to technology?",
      "answer": "B",
      "parameters": {
        "Entity1": "Microsoft",
        "Entity2": "Apple",
        "DomainX": "technology"
      }
    }
  ]
}
```

User: Are Ken Kwapis and Jan Švankmajer both filmmakers? Options: A. No B. Yes. Answer: B. Masked Question: Are Person1 and Person2 both ProfessionX? Parameters: 'Person1': 'Christopher Nolan', 'Person2': 'Quentin Tarantino', 'ProfessionX': 'filmmakers'.
Assistant:

```
{
  "similar_questions": [
    {
      "question": "Are Christopher Nolan and Quentin Tarantino both filmmakers?",
      "answer": "B",
      "parameters": {
        "Person1": "Christopher Nolan",
        "Person2": "Quentin Tarantino",
        "ProfessionX": "filmmakers"
      }
    },
    {
      "question": "Are J.K. Rowling and Stephen King both authors?",
      "answer": "B",
      "parameters": {
        "Person1": "J.K. Rowling",
        "Person2": "Stephen King",
        "ProfessionX": "authors"
      }
    },
    {
      "question": "Are Beyoncé and Elon Musk both singers?",
```

```
      "answer": "A",
      "parameters": {
        "Person1": "Beyoncé",
        "Person2": "Elon Musk",
        "ProfessionX": "singers"
      }
    },
    {
      "question": "Are Henry Ford and Albert Einstein both automobile entrepreneurs?",
      "answer": "A",
      "parameters": {
        "Person1": "Henry Ford",
        "Person2": "Albert Einstein",
        "ProfessionX": "automobile entrepreneurs"
      }
    },
    {
      "question": "Are Tony Hawk and Bob Burnquist both professional skateboarders?",
      "answer": "B",
      "parameters": {
        "Person1": "Tony Hawk",
        "Person2": "Bob Burnquist",
        "ProfessionX": "professional skateboarders"
      }
    }
  ]
}
```

User: Which of these is one similarity of the elements in the first group of the periodic table? Options: A. They all react similarly with water B. They have similar electron configurations. Answer: A. Masked Question: Which of these is one similarity of the elements in Group1 of the periodic table? Parameters: 'Group1': 'first group'.
Assistant:

```
{
  "similar_questions": [
    {
      "question": "Which of these is one similarity of the elements in the 18th group of the periodic table?",
      "answer": "B",
      "parameters": {
        "Group1": "18th group"
      }
    },
    {
      "question": "Which of these is one similarity of the elements in the alkali metals of the periodic table?",
      "answer": "B",
      "parameters": {
        "Group1": "alkali metals"
      }
    },
    {
      "question": "Which of these is one similarity of the elements in the noble gases of the periodic table?",
      "answer": "B",
      "parameters": {
        "Group1": "noble gases"
      }
    },
    {
      "question": "Which of these is one similarity of the elements in the halogens of the periodic table?",
      "answer": "A",
      "parameters": {
        "Group1": "halogens"
      }
    },
    {
      "question": "Which of these is one similarity of the elements in the transition metals of the periodic table?",
      "answer": "B",
      "parameters": {
        "Group1": "transition metals"
      }
    }
  ]
}
```

User: Which celestial object listed below has the greatest density? Options: A. a planet B. a neutron star. Answer: B. Masked Question: Which celestial object listed below has the Superlative Property? Parameters: 'Superlative': 'greatest', 'Property': 'density'.
Assistant:

```
{
  "similar_questions": [
    {
      "question": "Which celestial object listed below has the smallest density?",
      "answer": "A",
      "parameters": {
        "Superlative": "smallest",
        "Property": "density"
      }
    },
```

```
    {
      "question": "Which celestial object listed below has the highest temperature?",
      "answer": "A",
      "parameters": {
        "Superlative": "highest",
        "Property": "temperature"
      }
    },
    {
      "question": "Which celestial object listed below has the largest mass?",
      "answer": "B",
      "parameters": {
        "Superlative": "largest",
        "Property": "mass"
      }
    },
    {
      "question": "Which celestial object listed below has the brightest luminosity?",
      "answer": "B",
        "parameters": {
        "Superlative": "brightest",
        "Property": "luminosity"
      }
    },
    {
      "question": "Which celestial object listed below has the longest lifespan?",
      "answer": "B",
      "parameters": {
        "Superlative": "longest",
        "Property": "lifespan"
      }
    }
  ]
}
```

User: On what would you serve an egg to someone? Options: A. plate B. chicken coop. Answer: A. Masked Question: On what would you serve a FoodItem to someone? Parameters: 'FoodItem': 'egg'.
Assistant:

```
{
  "similar_questions": [
    {
      "question": "On what would you serve a cake to someone?",
      "answer": "A",
      "parameters": {
        "FoodItem": "cake"
      }
    },
    {
      "question": "On what would you serve a soup to someone?",
      "answer": "B",
      "parameters": {
        "FoodItem": "soup"
      }
    },
    {
      "question": "On what would you serve a salad to someone?",
      "answer": "B",
      "parameters": {
        "FoodItem": "salad"
      }
    },
    {
      "question": "On what would you serve a sandwich to someone?",
      "answer": "A",
      "parameters": {
        "FoodItem": "sandwich"
      }
    },
    {
      "question": "On what would you serve a steak to someone?",
      "answer": "B",
      "parameters": {
        "FoodItem": "steak"
      }
    }
  ]
}
```

User: Fighting inflation means being aware of many things, such as what with loans? Options: A. rising interest rates B. loss of money. Answer: A. Masked Question: Fighting EconomicIssue means being aware of many things, such as what with FinancialAspect? Parameters: 'EconomicIssue': 'inflation', 'FinancialAspect': 'loans'.
Assistant:

```
{
  "similar_questions": [
    {
```

```
      "question": "Fighting inflation means being aware of many things, such as what with loans?",
      "answer": "A",
      "parameters": {
        "EconomicIssue": "inflation",
        "FinancialAspect": "loans"
      }
    },
    {
      "question": "Fighting recession means being aware of many things, such as what with unemployment?",
      "answer": "B",
      "parameters": {
        "EconomicIssue": "recession",
        "FinancialAspect": "unemployment"
      }
    },
    {
      "question": "Fighting deflation means being aware of many things, such as what with prices?",
      "answer": "B",
      "parameters": {
        "EconomicIssue": "deflation",
        "FinancialAspect": "prices"
      }
    },
    {
      "question": "Fighting an economic crisis means being aware of many things, such as what with market volatility?",
      "answer": "A",
      "parameters": {
        "EconomicIssue": "economic crisis",
        "FinancialAspect": "market volatility"
      }
    },
    {
      "question": "Fighting a budget deficit means being aware of many things, such as what with government spending?",
      "answer": "B",
      "parameters": {
        "EconomicIssue": "budget deficit",
        "FinancialAspect": "government spending"
      }
    }
  ]
}
```

User: NOTE: Make sure the choices and masked question remain the same. Question: {question} Options: {options}. Answer: {answer}. Masked Question: {masked_question}. Parameters: {entities}.
Assistant:

A.1.4 Program Generation Prompt For SFT

Prompt for *Answer Function Implementation*

System: Implement the provided `answer(...)` function exactly as specified by the signature in Python. Return an integer corresponding to the index of the correct option in `Options`. Use the helper function `retrieve(question: str, answer_type: Type) -> Any` only for simple fact-checking. Allowed `answer_type` values: `int`, `float`, `list`, `str`, or `bool`. Do not embed reasoning logic inside `retrieve` calls—just simple fact-check questions. If a single `retrieve` question requires multi-step reasoning, decompose it into multiple simpler `retrieve` calls. Assume `retrieve` is implemented; do not define it. Include every input parameter in your `answer` function. Produce only the `answer` function—no extra code, comments, or imports beyond what's necessary.

User: Masked Question: Are Entity1 and Entity2 both related to DomainX? Parameters: `dict_keys(['Entity1', 'Entity2', 'DomainX'])`. Options: `['No', 'Yes']`. Function header: `def answer(Entity1: str, Entity2: str, DomainX: str) -> int`.
Assistant:

```
from typing import Any

def answer(Entity1: str, Entity2: str, DomainX: str) -> int:
    # First check Entity1's relationship
    entity1_related = retrieve(
        f"Is {Entity1} related to {DomainX}?", bool
    )
    # Then check Entity2's relationship
    entity2_related = retrieve(
        f"Is {Entity2} related to {DomainX}?", bool
    )
    # Both must be related for "Yes"
    if entity1_related and entity2_related:
        return 1
    else:
        return 0
```

User: Masked Question: Are both founders of CompanyX still involved in the company? Parameters: `dict_keys(['CompanyX'])`. Options: `['No', 'Yes']`. Function header: `def answer(CompanyX: str) -> int`.
Assistant:

```
from typing import Any, List

def answer(CompanyX: str) -> int:
    founders = retrieve(
```

```
        f"Who are the founders of {CompanyX}?", list
    )
    still_involved = []
    for founder in founders:
        involved = retrieve(
            f"Is {founder} still involved with {CompanyX}?", bool
        )
        still_involved.append(involved)
    if all(still_involved):
        return 1
    else:
        return 0
```

User: Masked Question: Which of these is one similarity of the elements in Group1 of the periodic table? Parameters: `dict_keys(['Group1'])`. Options: `['They all react similarly with water', 'They have similar electron configurations.']`. Function header: `def answer(Group1: str) -> int`.
Assistant:

```
from typing import Any, List

def answer(Group1: str) -> int:
    elements = retrieve(
        f"List the elements in {Group1} of the periodic table", list
    )
    valence_counts = []
    for el in elements:
        count = retrieve(
            f"How many valence electrons does {el} have?", int
        )
        valence_counts.append(count)
    if len(set(valence_counts)) == 1:
        return 1
    water_reactivities = []
    for el in elements:
        reactive = retrieve(
            f"Is {el} reactive with water?", bool
        )
        water_reactivities.append(reactive)
    if len(set(water_reactivities)) == 1:
        return 0
    return 0
```

User: Masked Question: Which celestial object listed below has the Superlative Property? Parameters: `dict_keys(['Superlative', 'Property'])`. Options: `['a planet', 'a neutron star']`. Function header: `def answer(Superlative: str, Property: str) -> int`.
Assistant:

```
from typing import Any, List

def answer(Superlative: str, Property: str) -> int:
    planet_list = retrieve(
        f"Which known planet has the {Superlative} {Property}?", list
    )
    planet_name = planet_list[0]
    planet_value = retrieve(
        f"What is the {Property} of {planet_name} in standard units?", float
    )
    neutron_list = retrieve(
        f"Which known neutron star has the {Superlative} {Property}?", list
    )
    neutron_name = neutron_list[0]
    neutron_value = retrieve(
        f"What is the {Property} of {neutron_name} in the same units?", float
    )
    is_max = retrieve(
        f"Does '{Superlative}' mean selecting the highest {Property}?", bool
    )
    if is_max:
        return 0 if planet_value > neutron_value else 1
    is_min = retrieve(
        f"Does '{Superlative}' mean selecting the lowest {Property}?", bool
    )
    if is_min:
        return 0 if planet_value < neutron_value else 1
    return 0
```

User: Masked Question: On what would you serve a FoodItem to someone? Parameters: `dict_keys(['FoodItem'])`. Options: `['plate', 'bowl']`. Function header: `def answer(FoodItem: str) -> int`.
Assistant:

```
from typing import Any

def answer(FoodItem: str) -> int:
    served_on_plate = retrieve(
        f"Is a plate a common item to serve {FoodItem}?", bool
    )
    served_on_bowl = retrieve(
        f"Is a bowl a common item to serve {FoodItem}?", bool
```

```
    )
    if served_on_plate:
        return 0
    else:
        return 1
```

User: Masked Question: Did HistoricalFigure have ethnicity closer to NativeGroup than ForeignGroup? Parameters: `dict_keys(['HistoricalFigure', 'NativeGroup', 'ForeignGroup'])`. Options: `['No', 'Yes']`. Function header: `def answer(HistoricalFigure: str, NativeGroup: str, ForeignGroup: str) -> int`.
Assistant:

```
from typing import Any

def answer(HistoricalFigure: str, NativeGroup: str, ForeignGroup: str) -> int:
    has_native = retrieve(
        f"Did {HistoricalFigure} have ancestry from {NativeGroup}?", bool
    )
    has_foreign = retrieve(
        f"Did {HistoricalFigure} have ancestry from {ForeignGroup}?", bool
    )
    if has_native and not has_foreign:
        return 1
    if has_foreign and not has_native:
        return 0
    more_native = retrieve(
        f"Did {HistoricalFigure} have a greater proportion of {NativeGroup} ancestry than {ForeignGroup}?", bool
    )
    if more_native:
        return 1
    else:
        return 0
```

User: Masked Question: Which reaction did Person have after TimePeriod in a ColdEnvironment? Parameters: `dict_keys(['Person','TimePeriod','ColdEnvironment','PhysicalReaction'])`. Options: `['chills','calm down']`. Function header: `def answer(Person: str, TimePeriod: str, ColdEnvironment: str, PhysicalReaction: str) -> int`.
Assistant:

```
from typing import Any

def answer(Person: str, TimePeriod: str, ColdEnvironment: str, PhysicalReaction: str) -> int:
    had_chills = retrieve(
        f"Did {Person} have chills after {TimePeriod} in a {ColdEnvironment}?", bool
    )
    if had_chills:
        return 0
    had_calm = retrieve(
        f"Did {Person} calm down after {TimePeriod} in a {ColdEnvironment}?", bool
    )
    if had_calm:
        return 1
    return 0
```

User: Masked Question: Which is a way that Earth loses energy from the EnergySource? Parameters: `dict_keys(['EnergySource'])`. Options: `['reflection','absorption']`. Function header: `def answer(EnergySource: str) -> int`.
Assistant:

```
from typing import Any

def answer(EnergySource: str) -> int:
    source_desc = retrieve(
        f"How does Earth lose energy from {EnergySource}?", str
    )
    is_reflection = retrieve(
        f"Does '{source_desc}' indicate reflection?", bool
    )
    if is_reflection:
        return 0
    is_absorption = retrieve(
        f"Does '{source_desc}' indicate absorption?", bool
    )
    if is_absorption:
        return 1
    return 0
```

User: NOTE: You must use all input parameters, decompose complex queries into multiple `retrieve` calls, and ensure at least two `retrieve` calls per function. {input}
Assistant:

A.1.5 Program Generation Prompt For RL

Prompt for *Answer Function Implementation*

System: Implement the provided `answer(...)` function exactly as specified by the signature in Python. Return an integer corresponding to the index of the correct option in `Options`. Use the helper function `retrieve(question: str, answer_type: Type) -> Any` only for simple fact-checking. Allowed `answer_type` values: `int`, `float`, `list`, `str`, or `bool`. Do not embed reasoning logic inside `retrieve` calls—just simple fact-check questions. If a single `retrieve` question requires multi-step reasoning, decompose it into multiple simpler `retrieve` calls. Assume `retrieve` is implemented; do not define it. Include every input parameter in your `answer` function. Produce only the `answer` function—no extra code, comments, or imports beyond what's necessary.

User: Masked Question: Are both founders of CompanyX still involved in the company? Parameters: `dict_keys(['CompanyX'])`. Options: `['No', 'Yes']`. Function header: `def answer(CompanyX: str) -> int`.
Assistant:

```
from typing import Any, List

def answer(CompanyX: str) -> int:
    founders = retrieve(
        f"Who are the founders of {CompanyX}?", list
    )
    still_involved = []
    for founder in founders:
        involved = retrieve(
            f"Is {founder} still involved with {CompanyX}?", bool
        )
        still_involved.append(involved)
    if all(still_involved):
        return 1
    else:
        return 0
```

User: Masked Question: Which celestial object listed below has the Superlative Property? Parameters: `dict_keys(['Superlative', 'Property'])`. Options: `['a planet', 'a neutron star']`. Function header: `def answer(Superlative: str, Property: str) -> int`.
Assistant:

```
from typing import Any, List

def answer(Superlative: str, Property: str) -> int:
    planet_list = retrieve(
        f"Which known planet has the {Superlative} {Property}?", list
    )
    planet_name = planet_list[0]
    planet_value = retrieve(
        f"What is the {Property} of {planet_name} in standard units?", float
    )
    neutron_list = retrieve(
        f"Which known neutron star has the {Superlative} {Property}?", list
    )
    neutron_name = neutron_list[0]
    neutron_value = retrieve(
        f"What is the {Property} of {neutron_name} in the same units?", float
    )
    is_max = retrieve(
        f"Does '{Superlative}' mean selecting the highest {Property}?", bool
    )
    if is_max:
        return 0 if planet_value > neutron_value else 1
    is_min = retrieve(
        f"Does '{Superlative}' mean selecting the lowest {Property}?", bool
    )
    if is_min:
        return 0 if planet_value < neutron_value else 1
    return 0
```

User: Masked Question: On what would you serve a FoodItem to someone? Parameters: `dict_keys(['FoodItem'])`. Options: `['plate', 'bowl']`. Function header: `def answer(FoodItem: str) -> int`.
Assistant:

```
from typing import Any

def answer(FoodItem: str) -> int:
    served_on_plate = retrieve(
        f"Is a plate a common item to serve {FoodItem}?", bool
    )
    served_on_bowl = retrieve(
        f"Is a bowl a common item to serve {FoodItem}?", bool
    )
    if served_on_plate:
        return 0
    else:
        return 1
```

User: Masked Question: Did HistoricalFigure have ethnicity closer to NativeGroup than ForeignGroup? Parameters: `dict_keys(['HistoricalFigure', 'NativeGroup', 'ForeignGroup'])`. Options: `['No', 'Yes']`. Function header: `def answer(HistoricalFigure: str, NativeGroup: str, ForeignGroup: str) -> int`.
Assistant:

```
from typing import Any

def answer(HistoricalFigure: str, NativeGroup: str, ForeignGroup: str) -> int:
    has_native = retrieve(
        f"Did {HistoricalFigure} have ancestry from {NativeGroup}?", bool
    )
    has_foreign = retrieve(
        f"Did {HistoricalFigure} have ancestry from {ForeignGroup}?", bool
    )
    if has_native and not has_foreign:
        return 1
    if has_foreign and not has_native:
        return 0
    more_native = retrieve(
        f"Did {HistoricalFigure} have a greater proportion of {NativeGroup} ancestry than {ForeignGroup}?", bool
    )
    if more_native:
        return 1
    else:
        return 0
```

User: NOTE: You must use all input parameters, decompose complex queries into multiple `retrieve` calls, and ensure at least two `retrieve` calls per function. `{input}`

A.1.6 Program Judge & Regeneration Prompt (RL)

Prompt for *Program Evaluation and Improvement*

System: You are given a group of questions derived from the same masked template. Your job is (1) to evaluate the PREVIOUS program and assign a single integer score, and (2) to REGENERATE one improved Python function that solves *all* questions in the group. Think step-by-step. Then output a valid JSON with exactly two keys: `{"score": <int>, "program": "<code>"}`.

Tasks (output JSON with "score" and "program"):
A) Evaluate the PREVIOUS program and produce ONE integer "score" (1–10) based on three dimensions:

1. **Factor alignment:** covers important reasoning factors from the Reasoning Path (e.g., alternatives, typical requirements, cultural variability, conservative defaults).
2. **No shortcuts:** no hard-coded mappings/dictionaries; no direct string checks for specific entities; no pattern-matching the literal question text; no label leakage.
3. **Proper decomposition:** breaks the task into simple, orthogonal subquestions via `retrieve()` or equivalent evidence checks that generalize to unseen items (not relying on concrete examples).

Scoring rubric (guidance):

- 9–10: strong factor coverage, no shortcuts, clear multi-step decomposition with robust fallbacks.
- 7–8: good factor coverage, minor gaps, mostly clean decomposition.
- 5–6: partial factor coverage and/or weak decomposition.
- 3–4: major gaps; some shortcut-like behavior or brittle logic.
- 1–2: fails most dimensions; relies on prohibited shortcuts or ignores factors.

B) Generate a NEW program as a single Python function:

- Signature: `def answer(..) -> int`. Return `0` or `1`.
- Must use decomposition into general, masked sub-queries (e.g., via `retrieve()`) that do not include concrete items or terms from the questions. Only use placeholders from inputs (e.g., `FoodItem`, `Utensil`) and generic concepts.
- Must not define or rely on any hard-coded mapping/dictionary/list of specific entities; must not read or reference the literal question text.
- Should reflect the Premise and the Inference. Keep the logic self-contained except for a black-box `retrieve(prompt: str, type_hint: type) -> Any`.
- Code should be concise, readable, and deterministic given `retrieve`'s returns; include brief comments.

OUTPUT FORMAT:

```
{
  "score": <integer 1-10>,
  "program": "<the improved Python code as a single string>"
}
```

User: Masked template

```
- Masked question: "<masked_question>"
- Choices: <choices>
- Options: ["A", "B"]


Questions and results of the previous program

- Q0: "<question_0>" | Gold: <gold_0> | Program: <pred_0>
- Q1: "<question_1>" | Gold: <gold_1> | Program: <pred_1>
...


Reasoning Path (reference facts you should respect)
<reasoning_path_text>


Previous candidate program (to be evaluated and then improved)
<previous_program_code_block>
```

A.2 RL training

We train our RL models on two H200 GPUs using the VERL framework (Sheng et al., 2025). The retriever runs with temperature 0.7. To characterize memory footprint, we co-locate an Instruct serving endpoint and training on the same two H200s: during backpropagation GPU memory saturates, whereas at inference it uses about half of the capacity. Unless stated otherwise, we use a learning rate of 1×10^{-5}, a KL-divergence coefficient of 0.001, five rollouts per update, a training batch size of 128, and rollout temperature $= 1$.

A.3 Example

Prompt messages = ["role": "system", "content": "Let's solve the following question step by step", "role": "user", "content": f"{question}",]

Result Question: Can you order a Tesla at Dunkin' Donuts?
Answer: Certainly! Letś break this down step by step:1. **Identify the Companies**: - Tesla is an electric vehicle manufacturer. - DunkinĎonuts is a coffee and donut chain.2. **Understand Their Business Models**: - Tesla sells vehicles, primarily electric cars, through its own dealerships and online. - DunkinĎonuts sells food and beverages in physical stores and through various digital platforms.3. **Check for Overlap**: - There is no direct business relationship or overlap between Tesla and DunkinĎonuts. They operate in completely different industries.4. **Consider the Question**: - The question asks if you can "order" a Tesla at DunkinĎonuts. This implies purchasing or ordering a product from one company at the location of another.5. **Conclusion**: - You cannot order a Tesla at DunkinĎonuts because Tesla is not a product sold by DunkinĎonuts, and there is no mechanism for ordering a Tesla through a DunkinĎonuts location. Therefore, the answer to the question "Can you order a Tesla at DunkinĎonuts?" is no.

Question: Can you order an Alfa Romeo at Starbucks?
Answer: Certainly, letś break this down step by step: 1. **Understanding the Context**: - An Alfa Romeo is a brand of luxury sports cars and sedans. - Starbucks is a global coffee chain known for its beverages, food items, and retail products.2. **Analyzing the Question**: - The question asks if one can "order" an Alfa Romeo at Starbucks. - This implies that the person wants to purchase or acquire an Alfa Romeo from Starbucks.3. **Evaluating Possibilities**: - Starbucks primarily sells beverages, food, and some merchandise related to their brand. - They do not sell vehicles, including cars like Alfa Romeos.4. **Conclusion**: - It is not possible to order an Alfa Romeo at Starbucks because they do not sell automobiles. Therefore, the answer to the question "Can you order an Alfa Romeo at Starbucks?" is no.

A.4 Reward Design

Our composite reward function R is designed to guide the model toward effective and generalizable reasoning. It is defined as $R = R_{\text{acc}} + R_{\text{ret}} + R_{\text{rej}} + R_{\text{fc}} + R_{\text{sa}}$. and comprises two families: *execution-based*

signals $\left(R_{\text{acc}}, R_{\text{ret}}, R_{\text{rej}}\right)$ that are directly verifiable from program execution, and *critique-based* signals $\left(R_{\text{fc}}, R_{\text{sa}}\right)$ that come from a frozen judge model.

- **Accuracy Reward (R_{acc}):** This reward is calculated as $R_{\text{acc}} = 0.2 \cdot n_{\text{correct}}$, where $n_{\text{correct}} \in \{0, 1, \ldots, 6\}$ is the number of correctly answered questions within the cohort of similar questions (1 original + 5 similar). This yields $R_{\text{acc}} \in [0, 1.2]$. We scale this term to 0.2 per correct so that gaining a few additional correct variants outweighs any single term, thereby aligning training with the cohort-consistency objective rather than stylistic surrogates.

- **Retrieve-Function Usage Reward (R_{ret}):** This component encourages appropriate problem decomposition and is assigned based on the number of retrieval calls (n_{calls}) made by the program:

$$R_{\text{ret}} = \begin{cases} -0.6 & \text{if } n_{\text{calls}} = 0 \\ 0 & \text{if } n_{\text{calls}} = 1 \\ +0.6 & \text{if } n_{\text{calls}} > 1 \end{cases}$$

  This results in $R_{\text{ret}} \in \{-0.6, 0, 0.6\}$. The retrieve-function usage reward encourages problem decomposition and discourages trivial solutions with no retrieval calls.

- **Rejection Penalty (R_{rej}):** This penalty discourages ineffective retrieval calls and is given by $R_{\text{rej}} = -0.1 \cdot n_{\text{rejected}}$, where $n_{\text{rejected}} \in \{0, 1, \ldots, 6\}$ is the number of questions in the group whose retrieve call is rejected by the retrieve model. This leads to $R_{\text{rej}} \in [-0.6, 0]$. The Rejection Penalty penalizes attempts to re-ask the original question or formulate overly similar questions, forcing multi-step, valid reasoning.

- **Factor-Complete Decomposition Reward(R_{fc}):** Given the program, the full set of similar questions, and failure patterns, a judging model scores whether the program's `retrieve` sub-questions cover the key factors needed to solve the original abstraction. Let $s_{\text{fc}} \in [1, 10]$ be this judged coverage score; we map it to $R_{\text{fc}} = 0.06 \cdot s_{\text{fc}}$, which yields $R_{\text{fc}} \in [0.06, 0.6]$. This term rewards factor-complete decompositions even when some variants remain wrong.

- **Structural Alignment Reward(R_{sa}):** Given the full context (cohort, original program p, failure patterns), a judge model proposes an improved program p^+. We then compare p and p^+ at the *logic level*: after stripping comments and formatting, we compute an AST-level structural similarity $s_{\text{logic}} \in [0, 1]$. We set $R_{\text{sa}} = 0.6 \cdot s_{\text{logic}} \in [0, 0.6]$.

  By construction, higher R_{sa} indicates that, after reflection, the logic of the original program p is becoming *closer* to that of its improved counterpart p^+—i.e., alignment in logic increases over training—rather than merely matching on superficial patterns.

The retrieve function reward is crucial for encouraging the model to break down complex problems into simpler, verifiable steps via appropriate information retrieval calls, rather than attempting to solve them monolithically or exploit loopholes (e.g., generating an empty program). The rejection mechanism explicitly blocks attempts to shortcut the reasoning process, such as re-asking the input question, thereby forcing genuine problem decomposition.

A.5 Sanity Check Experiment

Domain	Rejection	Upper-bound Acc
ARC-Challenge	72.8	96.0
ARC-Easy	79.7	96.0
CSQA	70.1	88.0
StrategyQA	65.7	92.0
HotpotQA	71.8	100.0
SimpleQA	46.0	—

Table 3: Unified sanity checks: Rejection rates(%) during RL training and estimated Upper-bound accuracies(%) after pass@128 sampling with manual program correction.

A.6 Ablation Study

A.6.1 Training on Original Questions Only

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
RL_{Org}	46.6 ± 4.4	55.4 ± 4.3	47.6 ± 4.4	36.8 ± 4.2	75.4 ± 3.8
$\text{RL}_{\text{Consistency}}$	$\mathbf{65.8 \pm 4.1}$	$\mathbf{74.8 \pm 3.8}$	$\mathbf{73.4 \pm 3.9}$	$\mathbf{45.8 \pm 4.4}$	$\mathbf{79.4 \pm 3.5}$

Table 4: Lenient accuracy (%) comparison between original (Org) and Consistency RL variant(Consistency). Bold = best

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
RL_{Org}	34.0 ± 4.1	42.2 ± 4.3	37.6 ± 4.2	20.8 ± 3.6	52.8 ± 4.4
$\text{RL}_{\text{Consistency}}$	$\mathbf{56.6 \pm 4.3}$	$\mathbf{68.0 \pm 4.1}$	$\mathbf{62.4 \pm 4.2}$	$\mathbf{31.0 \pm 4.0}$	$\mathbf{54.2 \pm 4.4}$

Table 5: Strict accuracy (%) comparison between original (Org) and Consistency RL variant(Consistency). Bold = best.

A.6.2 High-Quality Retriever & Judge

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
3B-3B	41.8 ± 4.3	40.8 ± 4.3	$\underline{42.6 \pm 4.3}$	$\mathbf{44.2 \pm 4.3}$	15.0 ± 3.1
3B-7B	$\underline{42.0 \pm 4.3}$	$\underline{41.0 \pm 4.3}$	42.2 ± 4.3	$\underline{44.0 \pm 4.3}$	$\underline{18.6 \pm 3.4}$
7B-7B	$\mathbf{56.8 \pm 4.3}$	$\mathbf{65.6 \pm 4.1}$	$\mathbf{65.6 \pm 4.1}$	39.2 ± 4.3	$\mathbf{23.6 \pm 4.2}$

Table 6: Lenient accuracy (%) across reasoning datasets with Qwen2.5-7B-Instruct serving as both retriever and judge for the 3B model. "3B-3B" refers to models trained and evaluated with the 3B retriever and judge; "3B-7B" refers to models trained with the 3B retriever/judge and evaluated with the 7B retriever; "7B-7B" refers to models trained and evaluated with the 7B retriever. **Bold** = best, underline = second best.

A.6.3 Out of Domain data Accuracy

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
3B-3B	$\underline{35.4 \pm 4.2}$	37.0 ± 4.2	$\underline{36.2 \pm 4.2}$	$\mathbf{32.6 \pm 4.1}$	7.6 ± 2.3
3B-7B	$\underline{35.4 \pm 4.2}$	$\underline{37.4 \pm 4.2}$	35.8 ± 4.2	$\mathbf{32.6 \pm 4.1}$	$\underline{11.4 \pm 2.8}$
7B-7B	$\mathbf{47.8 \pm 4.4}$	$\mathbf{56.4 \pm 4.3}$	$\mathbf{56.8 \pm 4.3}$	$\underline{25.8 \pm 3.8}$	$\mathbf{17.4 \pm 4.3}$

Table 7: Strict accuracy (%) across reasoning datasets with Qwen2.5-7B-Instruct serving as both the retriever and judge for the 3B model. Naming and notes follow Table 6. **Bold** = best, underline = second best.

Model	OpenBookQA	PubMedQA	MMLU
Vanilla	23.4 ± 3.6	31.2 ± 4.3	33.8 ± 3.7
$\text{SFT}_{\text{DM-5k}}$	57.6 ± 4.3	10.8 ± 2.7	40.0 ± 4.3
$\text{RL}_{\text{Normal}}$ (Exec+Crit)	$\mathbf{79.6 \pm 3.5}$	$\underline{41.2 \pm 4.3}$	$\underline{62.2 \pm 4.2}$
$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	$\underline{79.4 \pm 3.6}$	$\mathbf{50.0 \pm 4.4}$	$\mathbf{62.4 \pm 4.2}$

Table 8: Out-of-distribution (OOD) single-question accuracy (%) on OpenBookQA, PubMedQA, and MMLU. **Bold** = best, underline = second best.

A.7 CoT–Program

A.8 Human Analysis

A.9 Failure case

The following is a syntax failure case for $\text{RL}_{\text{Consistency}}$ ckpt.

Original Question: Do all of the African regions that participated in the Portugese Colonial War share an official language?

Maksed Question: Do all of the GeographicRegions that participated in the HistoricalEvent share an official Language?

Generated program:

```
from typing import List, Dict

def answer(GeographicRegions: str, HistoricalEvent: str, Language: str) -> int:
    # Split the input strings into individual regions
    regions = GeographicRegions.split(', ')

    # Check each region to see if it is associated with the Language
    all_regions_have_language = True
    for region in regions:
        region_participated = retrieve(
            f"Did {region} participate in the {HistoricalEvent}?",
            bool
        )
        region_official_language = retrieve(
            f"Is {Language} the official language of {region}?",
            bool
        )
        if region_participated and not region_official_language:
            all_regions_have_language = False
            break

    # If all regions are associated with the Language, return 'Yes'
    if all_regions_have_language:
```

	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
CoT strict accuracy	63.4	72.8	65.2	35.6	59.8
Program faithfulness audit (50 annotated pairs)					
Overall accept rate			34/50 = 68.0%		

Table 9: The CoT–program gap is real but bounded: CoT is informative on the underlying tasks, and a majority of annotated programs are judged by humans to be acceptable executable formalizations of the question.

Accuracy (%)		*Win Rate (%)*		
Answer Label	Abstraction Match	SFT	RL_{Normal}	$RL_{Consistency}$
92.0	96.7	23.0	30.0	47.0

Table 10: Human evaluation of generated questions and reasoning paths, reporting both accuracy metrics and win rates for different models.

```
        return 1
    else:
        return 0
```

The main issue here is that, at the beginning of the program, GeographicRegions is treated as a string containing many locations.

A.10 RL Hyperparameter Analysis

A.11 Single Question Accuracy

We also evaluate standard accuracy for the test set on 7B models as follows (See Table 14), which further shows the robustness of our method.

A.12 LLM Usage

During manuscript preparation, we used a general–purpose large language model (OpenAI ChatGPT) only for language polishing and structural editing. Concretely, we asked the model to: improve fluency and clarity at the sentence/paragraph level (grammar, wording, concision), suggest local reordering for better flow (e.g., merging redundant sentences, moving a definition earlier) and help finding related works. All edits were treated as suggestions and were reviewed, accepted, or rewritten by the authors.

Method	Ambiguity / Annotation	Control-Flow	Syntax	Retrieve
$\text{RL}_{\text{Consistency}}$	38	34	20	8
$\text{SFT}_{\text{DM-5k}}$	38	24	6	42

Table 11: Failure type distribution(%) on failure questions. RL reduces Retrieve errors while shifting residual errors to Control-Flow/Syntax.

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
Equal Reward	15.6 ± 3.2	18.2 ± 3.4	17.0 ± 3.3	2.6 ± 1.4	4.8 ± 1.9
2x Acc Reward	$\mathbf{35.4 \pm 4.2}$	$\mathbf{37.0 \pm 4.2}$	$\mathbf{36.2 \pm 4.2}$	$\mathbf{32.6 \pm 4.1}$	$\mathbf{7.6 \pm 2.3}$

Table 12: Strict accuracy (%) comparison between equal-weighted reward and double-accuracy reward for the 3B model. Bold = best.

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
Equal Reward	12.0 ± 2.9	13.8 ± 3.0	12.4 ± 2.9	0.8 ± 0.9	1.2 ± 1.0
2x Acc Reward	$\mathbf{41.8 \pm 4.3}$	$\mathbf{40.8 \pm 4.3}$	$\mathbf{42.6 \pm 4.3}$	$\mathbf{44.2 \pm 4.3}$	$\mathbf{15.0 \pm 3.1}$

Table 13: Lenient accuracy (%) comparison between equal-weighted reward and double-accuracy reward for the 3B model. Bold = best.

B Theory: Cohort Objectives, Robustness and Certification

This appendix formalizes the cohort-level objective used by our method and provides guarantees explaining why cohort-gated rewards better align with K-of-N consistency evaluation than instance-wise (mean) accuracy. We also model "shortcut" procedures and show that cohort gating suppresses them sharply. Finally, we provide a simple view of rejection penalties as a Lagrangian relaxation of a feasibility constraint and analyze the confidence bound for the method.

B.1 Setup and Notation

Let a denote a masked abstraction (template) and let a cohort $X = (x_1, \ldots, x_N)$ be drawn from a cohort distribution $\mathcal{D}(\cdot \mid a)$. A program p is produced by a stochastic policy $\pi_\theta(\cdot \mid a)$. When executed on a cohort instance x_i, the program may call a retrieval oracle and a rejection filter. We fold these components into the execution outcome.

Execution outcomes. Define

$$Z_i(p, X) \in \{0, 1\}$$

to indicate whether the program is counted correct on cohort member x_i under the evaluation protocol (including: wrong final answer, rejected retrieve call, runtime failure, etc. all count as 0). Define the cohort success count

$$S(p, X) := \sum_{i=1}^{N} Z_i(p, X).$$

For a threshold $K \in \{0, 1, \ldots, N\}$, define the cohort-consistency indicator

$$\mathbb{I}_K(p, X) := \mathbf{1}\{S(p, X) \geq K\}.$$

In the paper, $N = 6$ and $K \in \{4, 5\}$ correspond to lenient/strict criteria.

Policy-level objectives. We will compare two training surrogates:

Definition B.1 (Policy-level K-of-N objective)**.** The consistency objective is

$$J_K(\theta) := \mathbb{E}_a \, \mathbb{E}_{X \sim \mathcal{D}(\cdot|a)} \, \mathbb{E}_{p \sim \pi_\theta(\cdot|a)} \big[\mathbb{I}_K(p, X)\big].$$

Methods	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
Vanilla	49.8 ± 4.4	58.6 ± 4.3	52.0 ± 4.4	16.6 ± 3.3	28.4 ± 3.9
$\text{SFT}_{\text{DM-5k}}$	51.8 ± 4.4	61.8 ± 4.2	56.0 ± 4.3	24.2 ± 3.7	$\mathbf{29.0 \pm 4.0}$
$\text{RL}_{\text{Consistency}}$ (Exec+Crit)	$\mathbf{75.2 \pm 3.8}$	$\mathbf{85.0 \pm 3.1}$	$\mathbf{82.6 \pm 3.3}$	$\mathbf{44.2 \pm 4.3}$	28.6 ± 4.0

Table 14: Single Question accuracy across datasets for 7B model. Here $\pm$ denotes the half-width of the 95% Wilson confidence interval. Bold = best.

Model	ARC-Challenge	ARC-Easy	CSQA	StrategyQA	HotpotQA
K+1	35.6 ± 4.1	35.2 ± 4.4	$\mathbf{37.4 \pm 4.6}$	34.2 ± 3.9	$\mathbf{8.4 \pm 2.5}$
Normal K	$\mathbf{35.4 \pm 4.2}$	$\mathbf{37.0 \pm 4.2}$	36.2 ± 4.2	$\mathbf{32.6 \pm 4.1}$	7.6 ± 2.3

Table 15: Strict accuracy (%) comparison between k+1 and normal k for the 3B model. Bold = best.

Definition B.2 (Policy-level mean (normal) accuracy)**.** The mean-accuracy surrogate is

$$A(\theta) := \mathbb{E}_a \, \mathbb{E}_{X \sim \mathcal{D}(\cdot|a)} \, \mathbb{E}_{p \sim \pi_\theta(\cdot|a)} \Big[\frac{1}{N} \sum_{i=1}^{N} Z_i(p, X) \Big].$$

B.2 Cohort-Gated Rewards Directly Optimize the Evaluation Objective

Lemma B.3 (Unbiased estimator for $J_K(\theta)$)**.** *Let (a, X, p) be sampled by drawing a, then $X \sim \mathcal{D}(\cdot \mid a)$, then $p \sim \pi_\theta(\cdot \mid a)$. Then $\mathbb{I}_K(p, X)$ is an unbiased Monte Carlo estimator of $J_K(\theta)$:*

$$\mathbb{E}\big[\mathbb{I}_K(p, X)\big] = J_K(\theta).$$

Proof. Immediate by the tower property of expectation and Definition B.1. □

Lemma B.3 is the formal sense in which the *indicator* cohort reward aligns with the reported metric: if the training-time accuracy reward is chosen as $R_{\text{ind}}(p, X) = \mathbb{I}_K(p, X)$, then maximizing its expectation directly maximizes the K-of-N consistency objective $J_K(\theta)$.

B.3 Mean Accuracy Can Misrank Policies Under K-of-N Consistency

Even when both training and evaluation use the same cohorts, optimizing mean accuracy $A(\theta)$ can prefer a policy that fails the strict consistency criterion more often.

Proposition B.4 (Mean accuracy can misrank policies)**.** *Fix $N = 6$ and $K = 5$. There exist two policies π, π' such that*

$$A(\pi') > A(\pi) \quad \textit{but} \quad J_5(\pi') < J_5(\pi).$$

Proof. Construct two policies over deterministic programs, and suppress (a, X) since the construction is cohort-agnostic.

Let π always emit a program p that achieves $S(p, X) = 5$ on every cohort, hence $\mathbb{I}_5(p, X) = 1$ always. Then $A(\pi) = 5/6$ and $J_5(\pi) = 1$.

Let π' emit a program q_6 with $S(q_6, X) = 6$ with probability 0.9 and a program q_4 with $S(q_4, X) = 4$ with probability 0.1. Then

$$A(\pi') = 0.9 \cdot \frac{6}{6} + 0.1 \cdot \frac{4}{6} = \frac{5.8}{6} > \frac{5}{6} = A(\pi),$$

but

$$J_5(\pi') = 0.9 \cdot 1 + 0.1 \cdot 0 = 0.9 < 1 = J_5(\pi).$$

□

Proposition B.4 shows that mean accuracy can prefer policies that occasionally collapse under strict criteria, even when they raise the expected number of correct variants.

B.4 Amplification: Cohort Gating Sharply Penalizes Non-Robust Procedures

The next results quantify why a K-of-N objective suppresses brittle reasoning that fails on a nontrivial fraction of variants.

Conditional i.i.d. model. Assume that for a *fixed* program p and abstraction a, each variant outcome is conditionally i.i.d.:

$$Z_i \mid (p, a) \sim \text{Bernoulli}(\rho(p, a)), \quad \text{independently for } i = 1, \ldots, N.$$

Then $S \mid (p, a) \sim \text{Binomial}(N, \rho(p, a))$ and

$$\Pr[S \geq K \mid (p, a)] = \Pr\big[\text{Binomial}(N, \rho(p, a)) \geq K\big].$$

Let $\tau := K/N$.

Theorem B.5 (Hoeffding-style amplification around the threshold)**.** *Fix (p, a) under the conditional i.i.d. model above and let $\rho = \rho(p, a)$. For any $\gamma > 0$:*

$$\textit{(i)}\ \ \rho \leq \tau - \gamma \implies \Pr[S \geq K \mid (p, a)] \leq \exp\big(-2N\gamma^2\big),$$
$$\textit{(ii)}\ \ \rho \geq \tau + \gamma \implies \Pr[S \geq K \mid (p, a)] \geq 1 - \exp\big(-2N\gamma^2\big).$$

Proof. Write S/N as the empirical mean of i.i.d. Bernoulli variables. For (i), if $\rho \leq \tau - \gamma$, then

$$\Pr[S \geq K] = \Pr\left[\frac{S}{N} - \rho \geq \tau - \rho\right] \leq \Pr\left[\frac{S}{N} - \rho \geq \gamma\right] \leq \exp(-2N\gamma^2)$$

by Hoeffding's inequality. The second claim follows similarly by bounding $\Pr[S < K] = \Pr[S/N < \tau]$ and applying Hoeffding to the deviation $\rho - \tau \geq \gamma$. □

Interpretation. Mean accuracy is controlled by ρ directly, whereas the K-of-N event depends on whether ρ clears a *high threshold* τ. When τ is large (e.g., strict $K = 5$ of $N = 6$ gives $\tau \approx 0.833$), procedures with ρ even modestly below τ have exponentially small chance to pass the cohort criterion as N grows. This formalizes the idea that cohort gating suppresses brittle reasoning that fails on a non-negligible portion of variants.

Caveat. The conditional i.i.d. assumption is an idealization used to expose the thresholding effect cleanly. In practice, variant outcomes can be non-identically distributed and correlated, in which case the bound remains a qualitative explanation rather than a tight characterization.

B.5 A Simple Shortcut Model

We now model a "semantic shortcut" program that succeeds only when a spurious cue is present. This captures the intuition that instance-specific triggers may produce correct answers on some variants but do not generalize across the cohort.

Cue model. For a fixed abstraction a, assume each cohort member independently contains a spurious cue with probability $q \in [0, 1]$. Consider a shortcut program p_{sh} that succeeds iff the cue is present, so $\rho(p_{\text{sh}}, a) = q$ in the i.i.d. model above. Then:

$$\mathbb{E}\Big[\frac{S}{N} \mid (p_{\text{sh}}, a)\Big] = q \quad \text{but} \quad \Pr[S \geq K \mid (p_{\text{sh}}, a)] = \Pr[\text{Binomial}(N, q) \geq K].$$

Corollary B.6 (Shortcuts are sharply rejected when $q < \tau$)**.** *If $q \leq \tau - \gamma$ for some $\gamma > 0$, then*

$$\Pr[S \geq K \mid (p_{\text{sh}}, a)] \leq \exp(-2N\gamma^2),$$

even though the mean success rate of the shortcut is q.

Proof. Apply Theorem B.5 with $\rho = q$. □

Corollary B.6 formalizes why cohort gating discourages deceptive shortcuts: even if a shortcut yields decent mean accuracy, it is unlikely to satisfy a strict K-of-N requirement unless the cue appears in most variants.

B.6 Sensitivity of Strict Consistency to Weak Links

The strictest case $K = N$ (all variants must be correct) yields a clean closed form under conditional independence:

Lemma B.7 (Multiplicative sensitivity for $K = N$)**.** *Assume conditional independence for a fixed (p, a) but allow non-identical per-variant success probabilities $p_i = \Pr[Z_i = 1 \mid (p, a)]$. Then*

$$\Pr[S = N \mid (p, a)] = \prod_{i=1}^{N} p_i \;\le\; \Big(\frac{1}{N}\sum_{i=1}^{N} p_i\Big)^N$$

by AM–GM.

Proof. The first equality is independence. The inequality is AM–GM. □

Lemma B.7 states that strict cohort success is sensitive to even a single weak variant: one low p_i collapses the product.

B.7 Rejection Penalties as a Lagrangian Relaxation

Let $C(p, X) \ge 0$ be the number (or fraction) of rejected retrieve calls produced when executing p on cohort X. Consider a simplified objective that uses only an accuracy signal and a rejection penalty:

$$\max_{\theta} \;\; \mathbb{E}[\mathbb{I}_K(p, X)] - \lambda\,\mathbb{E}[C(p, X)] \quad \text{where} \quad p \sim \pi_\theta(\cdot \mid a).$$

This corresponds to maximizing a Lagrangian for a constrained problem.

Proposition B.8 (Penalty form corresponds to a constrained relaxation)**.** *Fix $\lambda > 0$. Any maximizer of*

$$\max_{\theta} \;\; \mathbb{E}[\mathbb{I}_K(p, X)] - \lambda\,\mathbb{E}[C(p, X)]$$

is Pareto-optimal for the bi-objective tradeoff between maximizing $J_K(\theta)$ and minimizing $\mathbb{E}[C]$. Moreover, for any constraint level c, there exists a λ (under mild regularity) such that a solution of the penalized problem is also a solution of

$$\max_{\theta} \;\; \mathbb{E}[\mathbb{I}_K(p, X)] \quad \textit{s.t.} \quad \mathbb{E}[C(p, X)] \le c.$$

Proof sketch. This is a standard Lagrangian relaxation statement: the penalized objective is the Lagrangian of the constrained problem. Under convexity/compactness assumptions on the achievable set of pairs $(\mathbb{E}[\mathbb{I}_K], \mathbb{E}[C])$, supporting hyperplanes correspond to some λ. Even without convexity, any maximizer is Pareto-optimal, since otherwise one could improve both terms and increase the penalized objective. □

Proposition B.8 provides a principled view: rejection penalties push the policy toward the feasible set of "atomic-only" tool usage without requiring hard constraints during optimization.

B.8 Certification View: Cohort Tests as Confidence Bounds on Per-Abstraction Reliability

A complementary interpretation of the cohort test is as a *statistical certificate* of per-abstraction reliability. Intuitively, a cohort $X = (x_1, \ldots, x_N) \sim \mathcal{D}(\cdot \mid a)$ provides N i.i.d. "trials" of the same underlying abstraction, so the observed success count $S(p, X)$ can be used to lower-bound the program's true success rate on new variants.

Per-abstraction reliability parameter. Fix an abstraction a and a program p. Under the conditional i.i.d. model from Section B.4, there exists a per-variant success probability

$$\rho(p, a) \;:=\; \Pr_{x \sim \mathcal{D}(\cdot \mid a)}\big[Z(p, x) = 1\big],$$

such that for $X \sim \mathcal{D}(\cdot \mid a)$ the cohort outcomes satisfy $Z_i(p, X) \overset{\text{i.i.d.}}{\sim} \mathrm{Bernoulli}(\rho(p, a))$ and hence

$$S(p, X) \mid (p, a) \;\sim\; \mathrm{Binomial}\big(N, \rho(p, a)\big).$$

(Here $Z(p, x)$ denotes the same evaluation outcome as $Z_i(p, X)$ but for a single instance; equivalently, one may take $\rho(p, a)$ to be the Bernoulli parameter in the model $Z_i \mid (p, a) \sim \mathrm{Bernoulli}(\rho(p, a))$ introduced earlier.)

Lower confidence bounds. Given an observed success count $S = s$ out of N, we can compute a one-sided lower confidence bound for $\rho(p, a)$ at confidence level $1 - \delta$.

Definition B.9 (One-sided lower confidence bound (Clopper–Pearson))**.** For $\delta \in (0, 1)$ and integers $s \in \{0, \ldots, N\}$, define the (exact) one-sided Clopper–Pearson lower bound

$$\underline{\rho}_{\mathrm{CP}}(s, N, \delta) := \begin{cases} 0, & s = 0, \\ F^{-1}_{\mathrm{Beta}(s,\, N-s+1)}(\delta), & s \geq 1, \end{cases}$$

where $F^{-1}_{\mathrm{Beta}(\alpha,\beta)}(\cdot)$ is the quantile function of a $\mathrm{Beta}(\alpha, \beta)$ distribution.

(Alternatively, one may use a Bayesian credible lower bound from a Beta prior $\mathrm{Beta}(\alpha_0, \beta_0)$, yielding $\underline{\rho}_{\mathrm{Bayes}}(s, N, \delta) = F^{-1}_{\mathrm{Beta}(s+\alpha_0,\, N-s+\beta_0)}(\delta)$. This is typically less conservative than Clopper–Pearson, but we state results for the exact bound for clarity.)

Theorem B.10 (Cohort outcomes yield a reliability certificate)**.** *Fix (p, a) and assume the conditional i.i.d. model so that $S \sim \mathrm{Binomial}(N, \rho(p, a))$. Then the Clopper–Pearson lower bound is a valid $(1 - \delta)$ one-sided confidence bound:*

$$\Pr\Big[\rho(p, a) \;\geq\; \underline{\rho}_{\mathrm{CP}}\big(S(p, X), N, \delta\big) \;\Big|\; (p, a)\Big] \;\geq\; 1 - \delta.$$

Moreover, $\underline{\rho}_{\mathrm{CP}}(s, N, \delta)$ is nondecreasing in s, so on any run where the cohort criterion passes ($S(p, X) \geq K$) we automatically have the deterministic bound

$$\underline{\rho}_{\mathrm{CP}}\big(S(p, X), N, \delta\big) \;\geq\; \underline{\rho}_{\mathrm{CP}}(K, N, \delta).$$

Equivalently, passing the K-of-N test can be reported as the certificate: "with confidence $1 - \delta$, the per-variant success rate is at least $\underline{\rho}_{\mathrm{CP}}(K, N, \delta)$."

Proof sketch. The Clopper–Pearson interval is obtained by inverting exact binomial tail tests. Concretely, for $s \geq 1$ the value $\underline{\rho}_{\mathrm{CP}}(s, N, \delta)$ is the unique ρ_0 such that

$$\Pr_{T \sim \mathrm{Binomial}(N, \rho_0)}[T \geq s] \;=\; \delta,$$

and binomial upper tails are monotone increasing in ρ_0. Therefore, for any true $\rho = \rho(p, a)$, the event $\rho < \underline{\rho}_{\mathrm{CP}}(S, N, \delta)$ implies that the observed tail event $T \geq S$ occurred with probability at most δ, which yields the stated coverage bound. Monotonicity in s follows from standard properties of Beta quantiles (or equivalently, from the nesting of inverted binomial tests). □

What strict/lenient thresholds certify. This view turns the cohort metric into an explicit robustness statement about new variants. For our setting $N = 6$, taking a conventional $\delta = 0.05$ (95% one-sided confidence) yields the approximate certificates

$$\underline{\rho}_{\mathrm{CP}}(4, 6, 0.05) \approx 0.27, \qquad \underline{\rho}_{\mathrm{CP}}(5, 6, 0.05) \approx 0.42, \qquad \underline{\rho}_{\mathrm{CP}}(6, 6, 0.05) \approx 0.61.$$

Using a simple Bayesian credible bound with a uniform prior $\mathrm{Beta}(1, 1)$ gives slightly less conservative numbers:

$$\underline{\rho}_{\mathrm{Bayes}}(4, 6, 0.05) \approx 0.34, \qquad \underline{\rho}_{\mathrm{Bayes}}(5, 6, 0.05) \approx 0.48, \qquad \underline{\rho}_{\mathrm{Bayes}}(6, 6, 0.05) \approx 0.65,$$

illustrating how a "unanimous" cohort ($6/6$ correct) can be reported as a clear certificate (e.g., "with 95% credibility, per-variant success rate ≥ 0.65" under the Beta posterior).

Designing K via a target certificate. The certificate perspective also suggests an interpretable way to set strict/lenient thresholds: choose a desired reliability floor $\rho_{\min}$ and confidence level $1 - \delta$, then set

$$K \;:=\; \min\Big\{k \in \{0, \ldots, N\} : \underline{\rho}_{\mathrm{CP}}(k, N, \delta) \geq \rho_{\min}\Big\}.$$

In this sense, K-of-N evaluation is a discretized certification rule: passing indicates statistically significant evidence that the program generalizes across the abstraction's variants.

Limitations. As with Theorem B.5, the certification statement relies on the conditional i.i.d. idealization; correlations or systematic heterogeneity across variants weaken the literal binomial model. Nonetheless, the confidence-bound view remains a useful conceptual bridge from cohort consistency to a familiar notion of robustness certification.