Correct but Unselectable: The Hidden Interface Tax in Multi-Candidate Reasoning

Qianli Ma[1,3] **Zhiqing Tang**[1,3,*] **Fanshuai Meng**[1,3]
Zhi Yao[2,3] **Hanshuai Cui**[2,3] **Weijia Jia**[3,4,*]

[1]Faculty of Arts and Science, Beijing Normal University, Zhuhai, China
[2]School of Artificial Intelligence, Beijing Normal University, Beijing, China
[3]Institute of AI and Future Networks, Beijing Normal University, Zhuhai, China
[4]Guangdong Key Lab of AI and Multi-Modal Data Processing,
Beijing Normal-Hong Kong Baptist University, Zhuhai, China

Abstract

Small language models deployed under cost, latency, or privacy constraints cannot simply switch to larger models when reasoning fails. Multi-candidate generate-then-rerank offers a natural test-time scaling route. However, more candidates do not always yield higher accuracy. We identify a previously overlooked bottleneck between generation and selection: a correct answer may appear in the candidate pool yet fail to be selected because it is incomplete, unparseable, or misaligned with the answer mode the fixed selector expects. We term this loss the *hidden interface tax* and formalize a *coverage–conversion decomposition* that distinguishes raw oracle coverage from *selectable coverage*, the subset of problems where a correct candidate can actually be consumed by the selector. We propose Compatibility-Aware Candidate Construction (CACC), an interface layer between the candidate source and the fixed selector that repairs fragments, aligns answer modes, and removes selector-confusing noise without retraining either component. On numeric reasoning (GSM8K, `competition_math`), CACC raises both oracle coverage and final accuracy. On GPQA Diamond, CACC alone improves final accuracy from 5.6% to 19.2% (+13.6 pp); combined with proposer-side strengthening it reaches 25.8% (+20.2 pp). On MMLU-Pro, CACC exposes a coverage–conversion gap: answer-mode-compatible construction raises oracle coverage, but final accuracy decreases by 0.85 pp under the original selector; adding proposer-side strengthening improves final accuracy.

1 Introduction

Small-model reasoning systems are deployed in local assistants, intranet question answering, code/math copilots, and edge offline applications. In these scenarios, model invocation cost, response latency, data privacy, and deployment controllability constrain system design more directly than a single benchmark score. Compared to large models, small models are better suited to such constrained deployments, but they are also more likely to produce incomplete reasoning, wrong final answers, format drift, and inconsistent answer modes on multiple-choice questions (MCQ). Multi-candidate generate-then-rerank has emerged as a natural test-time scaling route for these settings, where additional sampling under a fixed generator and selector is expected to raise accuracy.

However, increasing the number of candidates does not automatically yield higher accuracy. Added candidates may be fragments, partial solutions, unparseable strings, or wrong answers that nonetheless attract the verifier. On MCQ tasks, candidates must produce the benchmark-mandated choice

*Corresponding authors: Zhiqing Tang (`zhiqingtang@bnu.edu.cn`) and Weijia Jia (`jiawj@bnu.edu.cn`).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

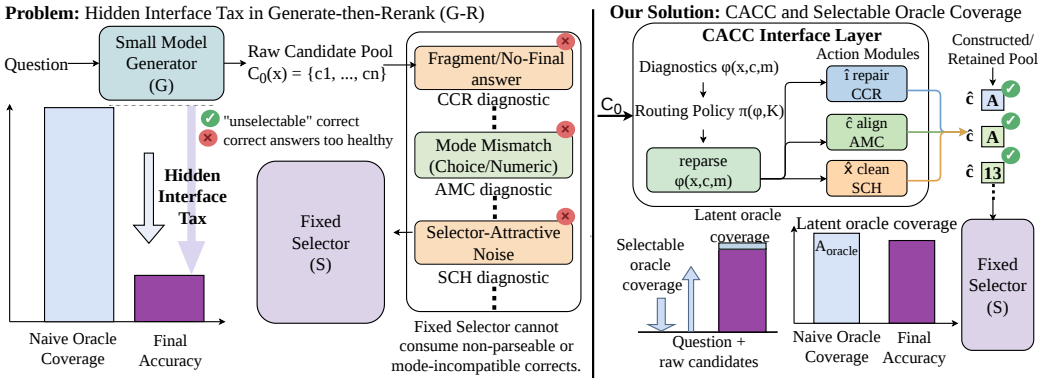

Figure 1: The missing candidate-interface layer in small-model multi-candidate reasoning. Existing methods optimize the candidate source or the selector, while whether the pool contains parseable, answer-mode-compatible, and selector-consumable candidates remains an independent bottleneck.

letter. Otherwise, correct underlying reasoning still fails at readout. These failure modes arise not at the candidate source or the selector, but at the interface between them (Figure 1). Existing test-time scaling and self-consistency methods focus on expanding the candidate source [1–3], while verifiers and rerankers focus on improving selection [4, 5], but neither addresses the candidate-interface layer that determines whether a correct candidate can actually be consumed by the fixed selector. The resulting loss from raw coverage to final accuracy constitutes a *hidden interface tax* that existing methods neither measure nor reduce. To address this, two challenges must be resolved.

The first challenge is how to identify and quantify the interface bottleneck separately from source quality and selector quality. Standard practice uses oracle coverage as the upper bound for final accuracy, measuring only whether any candidate in the pool is correct. Final accuracy alone conflates two distinct failure modes: the correct candidate is absent from the pool (a coverage failure), and the correct candidate is present but cannot be consumed by the fixed selector (a conversion failure). Without separating these, practitioners cannot diagnose whether to add more candidates or to fix interface alignment. Existing test-time scaling analyses track candidate count, sampling strategy, or reward-model quality [1–3, 6, 7], but do not isolate the interface as an independent bottleneck.

The second challenge is how to construct interface-compatible candidates under a fixed selector without accessing gold labels at test time. Interface failures span multiple types: incomplete reasoning before the final answer (fragments), answer-mode mismatch where a numeric answer appears in an MCQ slot or a choice letter is missing, and selector-confusing noise such as scaffold residue and near-duplicate wrappers. Trajectory-improvement methods such as Self-Refine and Reflexion alter the generation process itself [8, 9], and verifiers or process supervisors choose among candidates [4, 5, 10], but neither repairs the interface state of already-generated candidates. The construction mechanism must therefore operate between the fixed generator and the fixed selector, using only gold-free signals at inference time while drawing on offline-calibrated rules for routing and repair.

In this paper, we observe that the candidate interface between generation and selection is an independent, optimizable layer. We propose Compatibility-Aware Candidate Construction (CACC), which targets the distribution of candidates that reaches the fixed selector rather than improving the generator or the selector themselves. This layer-level separation explains why methods that raise oracle coverage sometimes fail to raise final accuracy, and why interface-side repair can succeed where further sampling cannot. On GPQA Diamond, CACC alone improves final accuracy from 5.6% to 19.2% (+13.6 pp).

The first two contributions address the measurement and construction challenges respectively. The main contributions are as follows.

1. **Coverage–conversion decomposition and selectable coverage.** We formalize the coverage–conversion gap in generate-then-rerank inference, defining *selectable coverage* to distinguish "the correct candidate has appeared" from "the correct candidate can be consumed and selected

by the fixed selector." This decomposition separates coverage and conversion failures, and relates selectable coverage to final accuracy under a fixed selector.

2. **Compatibility-Aware Candidate Construction (CACC).** We introduce CACC, an interface layer combining offline rule calibration with online gold-free candidate construction. The offline stage freezes a diagnostic rulebook from benchmark schemas, parser contracts, and audited failure cases. The online stage executes CCR (fragment repair), AMC (answer-mode alignment), and SCH (candidate hygiene) under the frozen policy, keeping the existing selector unchanged.
3. **Empirical effectiveness.** Under matched-budget settings, CACC raises both oracle and final accuracy on numeric reasoning (GSM8K, `competition_math`). On MCQ, composing CACC with proposer-side strengthening (SP) reaches $+20.2$ pp on GPQA Diamond and narrows the coverage–conversion gap on MMLU-Pro. Second-family paired closure on both benchmarks supports recoverable compatibility across generator families.

2 Related Work

Candidate-source expansion: multi-sampling, self-consistency, and search. Self-consistency, multi-sampling, Tree-of-Thought (ToT), and search-based reasoning enlarge candidate sources to raise the probability that a correct trajectory appears [1, 11–23]. Recent test-time scaling work analyzes compute allocation, reward model, and problem difficulty [2, 3, 6, 7, 24]. These methods expand source-side coverage but do not constrain whether candidates are complete, parseable, or answer-mode compatible, nor guarantee that a fixed verifier can select the correct candidate from the enlarged pool. Power Sampling shows that distribution shaping can raise raw oracle coverage yet hurt final accuracy when misaligned with the verifier, confirming that these are upstream extensions rather than same-layer alternatives to CACC.

Trajectory refinement and test-time feedback. Self-Refine, Reflexion, self-rewarding, code-aided reasoning, fine-tuning at test time (FTTT), and reasoning-model post-training improve trajectories via feedback, self-correction, or test-time optimization [8, 9, 25–34]. These methods alter the generation process or model state rather than the interface between existing candidates and the selector. CACC occupies a different position in the system stack, consuming outputs from the candidate source and performing fragment repair, answer-mode alignment, and candidate hygiene before the fixed selector.

Verifier/reranker and the candidate interface. Verifiers, rerankers, and preference optimization select the final answer from a candidate pool [4, 5, 35–42]. Recent work extends step-level checking, process supervision, thought reuse, and automated verifiers to math and general reasoning [10, 43–47]. However, a selector cannot rescue an oracle miss, and even when a correct candidate is present, candidate format, duplication, answer mode, and final-answer wrappers still alter selection behavior. The verifier's selection rate among oracle-hit candidates is therefore a joint outcome of the selector and the candidate interface, and CACC improves this input distribution without replacing the verifier.

Our experiments evaluate on numeric open-ended (GSM8K, `competition_math:numeric`) and MCQ (MMLU-Pro, GPQA Diamond) benchmarks [4, 48–51], contextualized against a broader set of reasoning benchmarks [52–57] and the open small-model ecosystem [31, 58–69]. None of these benchmarks or models includes an explicit candidate-interface evaluation, which is the gap CACC addresses.

3 Problem Formulation

This section formalizes the candidate interface in generate-then-rerank inference. We modify neither the small-model parameters nor the final selector. The focus is the interface state between the generator's outputs and the selector's inputs. The candidate source generates several candidates from the question, context, and possible options. The candidate-construction layer then repairs, aligns the answer mode, and cleans the raw candidates. Finally, the fixed selector picks one candidate from the retained pool, and a benchmark parser or scoring endpoint determines the final answer. Our central question is: under a fixed selector, can the candidate interface convert "the correct candidate has appeared" into "the correct candidate can be selected"?

For a sample in dataset $\mathcal{D}$, let x denote the input instance, y_x the ground-truth answer, and m the benchmark-locked answer mode (e.g. numeric or `choice_letter`). The generator first produces a raw candidate pool:

$$C_0(x) = \{c_1, \ldots, c_n\}. \tag{1}$$

The candidate-construction layer maps C_0 to a retained pool

$$C_K(x, m) = R_K(T_m(C_0(x), x)), \tag{2}$$

where T_m is a mode-aware candidate-construction operator and R_K controls deduplication, ranking, and the retained cap. The fixed selector S chooses

$$\hat{c} = S(x, C_K), \qquad \hat{y} = \mathrm{parse}_m(\hat{c}). \tag{3}$$

Here, $\mathrm{parse}_m(\cdot)$ denotes the benchmark-specific answer parser under answer mode m, e.g., numeric extraction for open-ended tasks and choice-letter extraction for MCQ tasks.

Our key constraint is the fixed selector. Within each Base/CACC comparison we modify only T_m or the candidate source, keeping the final selection function S unchanged.

We report four core quantities:

$$A_{\text{first}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{I}\{\mathrm{parse}_m(c_1) = y_x\}, \tag{4}$$

$$A_{\text{oracle}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{I}\{\exists c \in C_K(x, m) : \mathrm{parse}_m(c) = y_x\}, \tag{5}$$

$$A_{\text{final}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \mathbb{I}\{\mathrm{parse}_m(S(x, C_K)) = y_x\}, \tag{6}$$

$$A_{\text{verifier|oracle}} = \Pr[\mathrm{parse}_m(S(x, C_K)) = y_x \mid \exists c \in C_K(x, m) : \mathrm{parse}_m(c) = y_x]. \tag{7}$$

where $\mathbb{I}\{\cdot\}$ denotes the indicator function, which equals 1 when the condition holds and 0 otherwise. We abbreviate $A_{\text{verifier|oracle}}$ as V|O in tables and figures throughout.

For an extractive selector, $A_{\text{final}} = A_{\text{oracle}} \times A_{\text{verifier|oracle}}$ exactly, since a correct final answer requires a correct candidate in the pool. This decomposition separates two distinct bottlenecks: in a coverage-limited regime the pool lacks a correct answer, whereas in a compatibility-limited regime the correct answer is present but cannot be selected by the fixed selector.

Selectable coverage. Plain oracle coverage only requires that a parseable correct answer appear in the pool. *Selectable coverage* additionally requires that the correct candidate lie within a distribution the fixed selector can consume. Formally, define the *selectable-correct hit* event

$$H_{\text{sel}}(C_K, x, m) = \mathbb{I}\left\{ \begin{array}{c} \exists c \in C_K(x, m) : \ \mathrm{Correct}(c, y_x) \wedge \mathrm{Parseable}(c, m) \\ \wedge \ \mathrm{ModeMatch}(c, m) \wedge \mathrm{Clean}(c) \end{array} \right\}. \tag{8}$$

where $\mathrm{Correct}$ requires semantic correctness, $\mathrm{Parseable}$ requires the benchmark parser to read the candidate, $\mathrm{ModeMatch}$ requires the candidate to follow the answer mode m (a numeric span or `choice_letter`), and Clean requires the absence of selector-attractive noise such as instruction residue, scaffold leakage, or Invalid finals. This yields the *selectable oracle coverage*

$$A^{\text{sel}}_{\text{oracle}} = \Pr[H_{\text{sel}}(C_K, x, m) = 1]. \tag{9}$$

Let $h = A^{\text{sel}}_{\text{oracle}}$ and $q_b = \Pr[\mathrm{parse}_m(S(x, C_K)) = y_x \mid H_{\text{sel}} = b]$. The law of total probability gives the exact identity

$$A_{\text{final}} = hq_1 + (1 - h)q_0. \tag{10}$$

For a zero-probability branch, its weighted contribution is defined as zero. Writing $r = (1 - h)q_0$ and $A^{\text{sel}}_{\text{verifier|oracle}} = q_1$, we obtain

$$A_{\text{oracle}} - A_{\text{final}} = (A_{\text{oracle}} - h) + h(1 - q_1) - r. \tag{11}$$

When $r = 0$, h is guaranteed to be an upper bound; with a nonzero residual it need not be: an imperfect diagnostic predicate can reject a candidate that the selector nevertheless answers correctly. For an extractive selector with a common parser, raw oracle coverage remains an upper bound. The compatibility predicate is an operational diagnostic, while the residual accounts for its incomplete agreement with selection. We report oracle coverage, conditional conversion, and final accuracy directly; paired interventions in Section 5.1 measure specific interface effects.

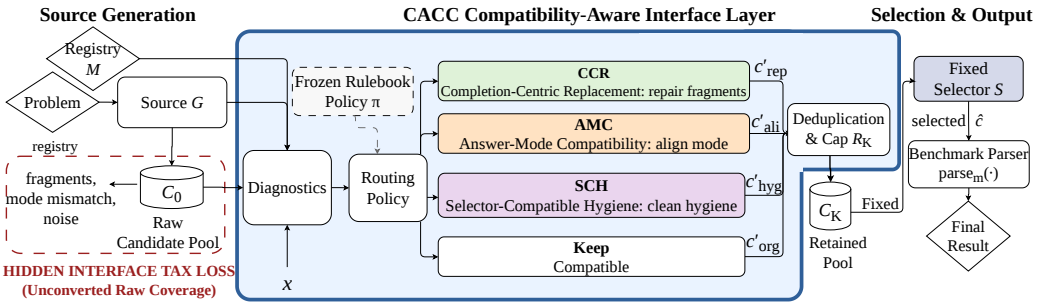

Figure 2: The offline–online flow of CACC. Lower half: offline calibration produces a frozen rulebook and routing policy. Upper half: online inference applies gold-free diagnostics and construction to the raw candidate pool before the fixed selector.

Worked example. On MMLU-Pro example `mmlu-pro-test-00072` (gold J), the recorded eight-candidate CACC pool parses to $[-, A, -, A, -, A, J, F]$. Candidate 7 contains `Final answer: J`, so oracle coverage is satisfied; majority readout selects A (recorded slot 2), so final correctness is zero. This post-hoc pool example makes the coverage–conversion distinction directly checkable.

4 Method

CACC is a two-stage candidate-interface system (Figure 2). The offline stage freezes a diagnostic rulebook from benchmark schemas and audited failure cases. The online stage applies the frozen rules to repair, align, and clean the raw candidate pool before it reaches the fixed selector. The coverage–conversion decomposition (Section 3) identifies where the interface bottleneck lies, and the construction pipeline described here targets it. The basic composition is

$$\text{CACC} = \text{CCR} + \text{AMC} + \text{SCH}. \tag{12}$$

Offline calibration. The offline module (lower half of Figure 2) takes benchmark schemas, parser configurations, prompt/candidate traces, scoring manifests, and a small set of audited failure cases as input, and outputs a frozen rulebook $\mathcal{R}$ and routing policy π. Failure attribution and threshold calibration may use dev or audit-split labels in this stage, but online inference reads no gold labels and never modifies rules according to test-set labels. Concretely, the offline stage first builds the answer-mode registry $\mathcal{M}$ and parser contract. It then defines candidate diagnostic features $\phi(x, c, m)$ covering parseability, completion, Answer-mode mismatch, wrapper consistency, residue, scaffold leakage, and selector-attractive noise. Next, it instantiates action rule templates for CCR/AMC/SCH. Finally, it calibrates the routing policy $\pi : \phi(x, c, m) \mapsto a \in \{\text{keep, repair, reparse, drop, replace}\}$ on the audit split. The MMLU-Pro `CCR-only`/no-AMC provenance in Table 3 shows that, without locking the answer-mode registry and parser contract, MCQ candidates are severely underestimated under an incorrect numeric readout; AMC is therefore a benchmark contract written into the rulebook offline, rather than an after-the-fact formatting step.

Online inference. The online module (upper half of Figure 2) samples a raw candidate pool C_0 under a fixed budget, computes gold-free observable signals ϕ (parser status, answer mode, final-answer marker, residue/wrapper, conflict, verifier score/margin) for each candidate, and routes them by the frozen policy π to buckets such as incomplete/no-final, answer-mode/parser mismatch, and residue/conflict/verifier-distractor; CCR/AMC/SCH actions are then applied per bucket. The system forms a union of original and constructed candidates, then applies deduplication, retained-cap selection, and a parser check to obtain C_K, which is finally passed to the unchanged fixed selector S. The three modules differ in where they intervene: CCR primarily affects coverage (when correct answers are missing), AMC primarily affects conversion (when candidates are incompatible with the answer mode), and SCH primarily improves selector input distribution (when correct candidates are drowned by noise and duplication). Algorithm 1 states the full policy-parameterized procedure. The offline rulebook components used in our implementation are summarized in Appendix Table 13.

Implementation details. In our reference implementation the routing policy π is a deterministic decision list parameterized by the diagnostic vector ϕ and a fixed retained-cap budget K. CCR fires

Algorithm 1 Offline–online CACC

Input: Benchmark schemas, parser configs, audited traces (offline); problem x, raw pool C_0, retained cap K, fixed selector S (online)
Output: Selected answer $\hat{y}$, retained pool C_K

// Offline calibration
Build answer-mode registry $\mathcal{M}$ and diagnostics ϕ
Instantiate rulebook $\mathcal{R} = \{\text{CCR-rules}, \text{AMC-rules}, \text{SCH-rules}\}$
Calibrate and freeze routing policy $\pi(\phi, K)$ on audited traces
// Online inference
Infer $m \leftarrow \mathcal{M}(x)$; diagnose $D = \{\phi(x, c, m) : c \in C_0\}$
Apply frozen rules: $C' \leftarrow \text{apply}(C_0, D, \pi, \mathcal{R})$
Retain: $C_K \leftarrow \text{retain}_K(\text{dedup}(\text{parsecheck}(C', m)), K)$
Select $\hat{c} \leftarrow S(x, C_K)$ {Selector unchanged}
Parse $\hat{y} \leftarrow \text{parse}_m(\hat{c})$
return $(\hat{y}, C_K)$

when at least one in-pool candidate carries a fragment or missing-final span and at least one slot falls below the parser's completion threshold, using completion-oriented generation before merging and fixed-cap retention. AMC fires whenever the answer-mode registry $\mathcal{M}(x)$ disagrees with a candidate's final-answer wrapper; for MCQ tasks it enforces `choice_letter` normalization on top of any non-letter answer, while for numeric tasks it normalizes the final numeric span and protects boxed-answer wrappers. SCH fires when residue, scaffold leakage, or Wrapper drift triggers the verifier's distractor signal; deduplication is applied last so that the retained cap is filled with mode-aligned, maximally diverse candidates rather than near-duplicates of a dominant scaffold. CCR, AMC, and SCH use deterministic rules (regex matching, span extraction, template filling); completion additionally calls the generator and requires rescoring. CACC keeps the retained cap fixed without retraining the verifier, but total inference cost is not matched. The full rulebook components and parser-binding contracts are reported in Appendix D and Appendix Table 16; completion settings and costs are in Appendix G.

5 Experiments and Mechanistic Analysis

Setup. All main experiments follow a generate-then-rerank protocol. Each comparison holds its retained cap, parser, scoring endpoint, and downstream selector fixed. The evidence spans three categories. Numeric open-ended reasoning (GSM8K / `competition_math:numeric`) evaluates direct final gains. Interface stress tests (MMLU-Pro / GPQA Diamond) probe MCQ answer-mode alignment, candidate hygiene, and bounded generator-family recovery. Composition evidence asks whether the coverage–conversion gap can be systematically changed by proposer-side strengthening, conversion-aware rescue, and coverage-pressure interventions.

Models and inference stack. The main pipeline uses Qwen3-1.7B to generate the raw candidate pool, CACC to process the candidate interface, and a trained verifier-reranker held fixed across comparisons to select the final answer. CACC-only and matched baselines share the same generator, retained cap, parser, and verifier. The CACC+SP rows use a Qwen3-8B proposer / prompt-mixture as proposer-side strengthening and are reported only as selected-regime composition evidence rather than folded into the canonical CACC result. Full checkpoint ids are listed in Appendix A.

Table 1 reports unified method rows for Base, SP, CACC, and CACC+SP across four datasets. SP denotes proposer-side strengthening with a Qwen3-8B proposer, a different compute regime from the Qwen3-1.7B generator used by Base and CACC. For numeric tasks, Base is the raw generate-then-rerank pool. For MMLU-Pro, Base is a CCR-only legacy pool scored under benchmark-locked MCQ reparse (see Appendix B). Final = Oracle $\times$ V|O, and Ratio is Final divided by the Base-row Final on the same dataset. The MMLU-Pro no-AMC archived numeric provenance (oracle 0.0369, final 0.0096) is used only for AMC necessity; see Appendix B.

Numeric reasoning and coverage–conversion decomposition. The dominant failure modes on numeric open-ended tasks are fragments, partial solutions, and missing-final answers. The numeric block of Table 1 together with Panels A–B of Figure 3 shows: GSM8K clean oracle 0.4882 $\rightarrow$

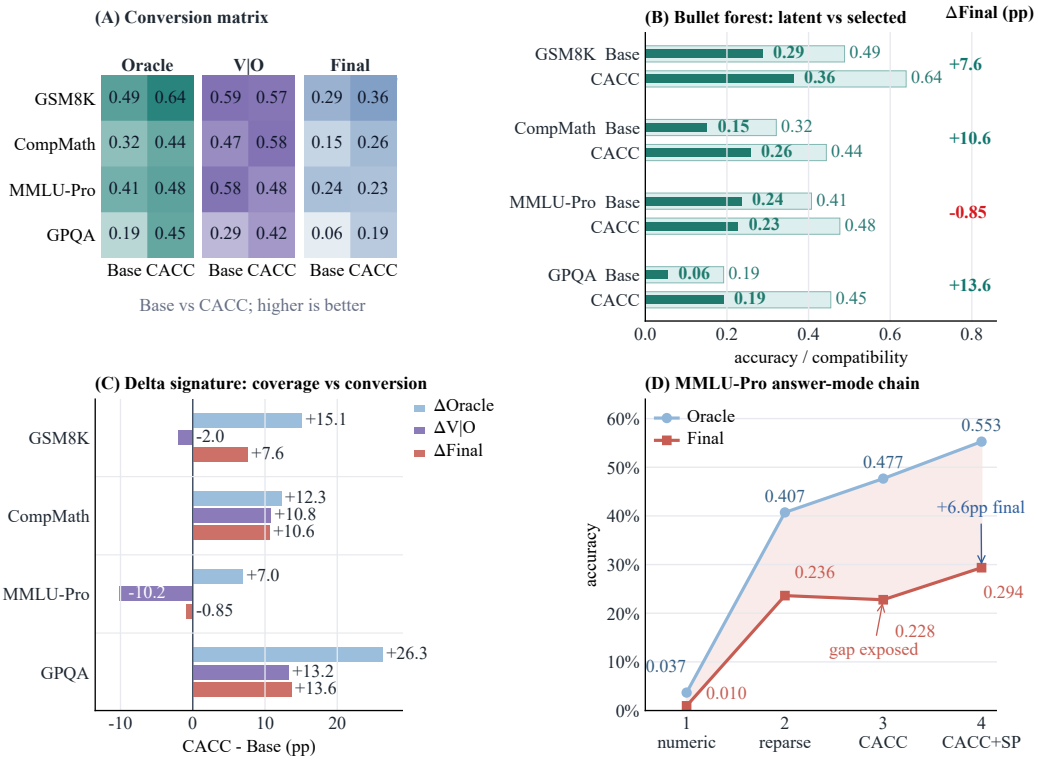

Figure 3: Main results across reasoning regimes. Panel A: Base/CACC oracle, V|O, and final. Panel B: latent coverage upper bound and selected final accuracy with CACC-vs-Base delta. Panel C: Δoracle, ΔV|O, and Δfinal separating coverage, conversion, and final. Panel D: four-step MMLU-Pro mechanism chain. Exact numbers are in Table 1 and Appendix Tables 2, 3.

0.6391 and final 0.2873 $\rightarrow$ 0.3632; `competition_math:numeric` oracle 0.3210 $\rightarrow$ 0.4436 and final 0.1519 $\rightarrow$ 0.2579. Under a fixed selector, CCR's repair of incomplete reasoning and missing-final answers improves both candidate coverage and final selection. However, coverage, V|O, and final do not move in lockstep (Table 2): the answer-mode interface can raise oracle and final together. Once readout is aligned, micro-ablations yield small marginal gains. a *diverse pool* $\rightarrow$ CACC transition can lower final when oracle rises but V|O drops, and adding hygiene then recovers final via V|O; on GSM8K full, SP $\rightarrow$ Power Sampling (PS)$\times$CACC shows raw oracle +10.84pp can still come with V|O and final drops. We therefore report the coverage–conversion decomposition rather than a single oracle/final indicator. In the numeric regime, CACC's gains transfer cleanly because the numeric answer mode imposes a lightweight readout contract. CCR's repair of fragments and missing-final answers maps directly to selectable coverage. The MMLU-Pro micro-ablation in Table 2 (CCR+AMC$\rightarrow$CACC, $\Delta F = +0.07$ pp) shows a small marginal gain from hygiene in this setting.

MCQ readout chain and second-family closure. What matters for MCQ is not the sheer number of candidates but the supply of answer-mode-compatible candidates that the fixed selector can consume. MCQ tasks impose stronger readout constraints on the candidate interface (Figure 3 Panel D). Under archived numeric scoring, the legacy/no-AMC candidate interface yields a final of only 0.0096. Switching to benchmark-locked choice-letter readout, the same-family CCR-only baseline recovers oracle/final to 0.4072/0.2362 (Appendix B). CACC further raises oracle to 0.4767 but holds final at 0.2277, exposing the coverage–conversion gap, and CACC+SP pushes final to 0.2936 in the selected regime.

GPQA Diamond exhibits the same direction with a more pronounced effect. Under the fixed-verifier interface, the Base final is only 0.0556 partly because answer-mode alignment is missing and many candidates are rejected by the readout as invalid (the predicted consequence of the hidden interface

Table 1: Candidate-interface results. ΔFinal and Ratio are computed against the Base row within the same dataset. † uses a Qwen3-8B proposer (different compute regime from the Qwen3-1.7B generator). Paired tests for CACC vs Base are in Appendix Table 10.

Regime	Dataset	Method	Oracle	V|O	Final	ΔFinal	Ratio
Numeric	GSM8K	Base	0.4882	0.5885	0.2873	ref.	1.00×
Numeric	GSM8K	SP†	0.6471	0.5714	0.3698	+8.25pp	1.29×
Numeric	GSM8K	CACC	0.6391	0.5682	0.3632	+7.59pp	1.26×
Numeric	GSM8K	CACC+SP†	0.6962	0.6017	0.4189	+13.16pp	1.46×
Numeric	CompMath	Base	0.3210	0.4732	0.1519	ref.	1.00×
Numeric	CompMath	SP†	0.4616	0.6014	0.2776	+12.57pp	1.83×
Numeric	CompMath	CACC	0.4436	0.5814	0.2579	+10.60pp	1.70×
Numeric	CompMath	CACC+SP†	0.5024	0.6173	0.3101	+15.82pp	2.04×
MCQ	MMLU-Pro	Base	0.4072	0.5800	0.2362	ref.	1.00×
MCQ	MMLU-Pro	SP†	0.4921	0.5411	0.2663	+3.01pp	1.13×
MCQ	MMLU-Pro	CACC	0.4767	0.4777	0.2277	-0.85pp	0.96×
MCQ	MMLU-Pro	CACC+SP†	0.5527	0.5312	0.2936	+5.74pp	1.24×
MCQ	GPQA	Base	0.1919	0.2897	0.0556	ref.	1.00×
MCQ	GPQA	SP†	0.4724	0.4197	0.1983	+14.27pp	3.57×
MCQ	GPQA	CACC	0.4545	0.4222	0.1919	+13.63pp	3.45×
MCQ	GPQA	CACC+SP†	0.4899	0.5258	0.2576	+20.20pp	4.63×

Table 2: Coverage–selection decomposition on audit rows (outside Table 1). O, V, F denote oracle, V|O, and final; b/a denote before/after; Δ in percentage points.

Unit	Comparison	O_b	O_a	V_b	V_a	F_b	F_a	ΔO	ΔV	ΔF
MMLU-Pro	no-AMC numeric → CACC	0.0369	0.4767	0.2590	0.4777	0.0096	0.2277	+43.98	+21.87	+21.81
MMLU-Pro	CCR+AMC → CACC	0.4775	0.4767	0.4754	0.4777	0.2270	0.2277	-0.08	+0.23	+0.07
MMLU-512	diverse → CACC	0.4609	0.5547	0.4492	0.3592	0.2070	0.1992	+9.38	-9.00	-0.78
MMLU-512	CACC → +hygiene	0.5547	0.5547	0.3592	0.4049	0.1992	0.2246	+0.00	+4.58	+2.54
GSM8K full	SP → PS x CACC	0.6111	0.7195	0.5943	0.4700	0.3632	0.3381	+10.84	-12.43	-2.50

tax). CACC-only raises final to 0.1919 ($+13.6$pp absolute gain), and CACC+SP reaches 0.2576 ($+20.2$pp). Second-family paired closure further supports recoverable compatibility: on MMLU-Pro confirm-512, CACC→+hygiene moves $0.1992 \to 0.2246$ (McNemar $p = 0.0072$); on GPQA-128, $0.2031 \to 0.2656$ ($p = 0.0215$). The Mistral-7B Smoke rows trend in the same direction (Table 3 and Appendix Table 10). The MMLU-Pro coverage–conversion gap is therefore not a failure mode of CACC but an explicit revelation. Under a fixed verifier and a strict choice-letter readout, raising oracle without aligning the answer-mode contract converts only after proposer-side strengthening reshapes the candidate distribution toward verifier-friendly options. GPQA Diamond shows the same pattern at higher amplitude precisely because its base is more strongly limited by readout invalidity, which is also why second-family hygiene (rather than a transplanted rulebook) recovers selectable coverage when the proposer changes.

Layer-aware positioning and gap-narrowing interactions. CACC should be understood relative to external enhancements at the system-layer level. Self-Refine improves trajectories, ToT/search expands the reasoning space, self-consistency expands the candidate source, FTTT applies fine-tuning at test time, and verifiers/rerankers change selection. CACC improves the candidate-interface distribution before any of these sources reaches the selector (per-method compute/budget/layer differences are listed in Appendix Table 12). The coverage–conversion gap can be narrowed jointly across these layers (Table 3 and Appendix Figure 4): on the proposer side, CACC+SP raises final from $0.2277/0.1919$ to $0.2936/0.2576$ on MMLU-Pro and GPQA respectively. Layering conversion-aware rescue further improves selection-side conversion (MMLU-Pro $0.2936 \to 0.3320$, $p = 3.43 \times 10^{-67}$; GPQA $0.2576 \to 0.2929$, $p = 0.0923$). Composability across layers is constructive but not additive: CACC+SP narrows the gap from the source side, conversion-aware rescue narrows it from the selection side, and the Power Sampling×CACC diagnostic in Table 3 shows that distribution shaping at the wrong layer can widen rather than narrow the gap. Layer matching (not the number of stacked enhancements) is what predicts whether a given composition converts oracle expansion into final accuracy under a fixed selector.

Table 3: Ablation, recovery, and interaction evidence. Reports oracle, V|O, and final before→after with the final delta; sample sizes, wins/losses, p-values, scope, and artifact statuses are in Appendix Tables 10, 11, and 12.

Type	Dataset	Method / transition	Oracle	V|O	Final	ΔF
READOUT	MMLU-Pro	numeric scoring → MCQ reparse	0.0369→0.4072	0.2590→0.5800	0.0096→0.2362	+22.66pp
AMC	MMLU-Pro	MCQ reparse → CACC	0.4072→0.4767	0.5800→0.4777	0.2362→0.2277	-0.85pp
SP	MMLU-Pro	CACC → CACC+SP	0.4767→0.5527	0.4777→0.5312	0.2277→0.2936	+6.59pp
SP	GPQA	CACC → CACC+SP	0.4545→0.4899	0.4222→0.5258	0.1919→0.2576	+6.57pp
RECOVERY	MMLU-Pro	CACC → +hygiene	0.5547→0.5547	0.3592→0.4049	0.1992→0.2246	+2.54pp
RECOVERY	GPQA	CACC → +hygiene	0.5127→0.4931	0.3961→0.5386	0.2031→0.2656	+6.25pp
RESCUE	MMLU-Pro	CACC+SP → +rescue	0.5527→0.5527	0.5312→0.6007	0.2936→0.3320	+3.84pp
RESCUE	GPQA	CACC+SP → +rescue	0.4899→0.4899	0.5258→0.5979	0.2576→0.2929	+3.53pp
DIAGNOSTIC	GSM8K	SP → PS × CACC	0.6111→0.7195	0.5943→0.4700	0.3632→0.3381	-2.50pp
SMOKE	GSM8K	Mistral diverse → selector-cal	0.3896→0.4179	0.5413→0.5422	0.2109→0.2266	+1.56pp
SMOKE	MMLU-Pro	Mistral diverse → filter+hygiene	0.4129→0.4367	0.4541→0.4829	0.1875→0.2109	+2.34pp

Table 4: Additional controls. Final accuracies are percentages; gains are percentage points. Frozen-pool rows reuse the original candidates. Target-verifier rows are three-seed means; Mistral is a five-seed mean. CD uses fresh paired generation and a common canonical selector, with its own baseline.

Control	Evaluation	N	Base	CACC	ΔF
Target verifier, canonical	GPQA	198	16.16	24.92	+8.75
Target verifier, canonical	MMLU held-out	11,520	26.40	27.84	+1.44
32B candidate judge	GPQA	198	6.57	23.23	+16.67
Mistral generator	MMLU confirm	512	14.22	15.94	+1.72
Prospective ARC reuse	ARC-Challenge	256	69.92	70.70	+0.78
Fresh generation	**Evaluation**	N	**CACC**	**CD**	**CD+CACC**
Paired four-cell control	GPQA	198	23.23	29.80	28.79
Paired four-cell control	MMLU confirm	512	29.30	30.27	28.12

5.1 Additional Controls and Prospective Validation

Table 4 adds the key controls; protocols and paired statistics are in Appendix G.

Trained verifiers and selector families. With non-evaluation supervision and canonical choice rendering, all three target-verifier seeds improve on GPQA and held-out MMLU-Pro. On frozen GPQA pools, majority voting, 8B option-logprob, and 1.7B/32B judges give gains of 3.03, 10.61, 13.13, and 16.67 pp. MMLU effects depend on selector/readout and scope. Mistral generation improves on all five seeds, with a mean gain of 1.72 pp.

Constrained decoding and format. The fresh four-cell control shares Qwen3-1.7B stochastic prefixes and a target-specific canonical selector. CACC improves final accuracy over Base by 6.64 pp on MMLU confirm-512. CD exceeds CACC by 6.57 pp on GPQA and 0.98 pp on MMLU confirm-512. CD+CACC adds coverage without improving final accuracy over CD. Content-preserving final-marker changes flip 9.89% of eligible MMLU selections, with little change in aggregate accuracy.

MMLU conversion and new-benchmark reuse. The MMLU-Pro majority-vote analysis shows limited conversion of newly added coverage and losses of previously available correct candidates. On ARC-Challenge, reusing the unchanged rules increases oracle coverage with little change in final accuracy (Appendix G.7).

Qualitative cases. The error taxonomy in Appendix Table 14 aligns with concrete audit examples (Appendix Table 15). On `gsm8k-test-00462`, the original Qwen3-1.7B candidate stops at a fragmented arithmetic step. CCR repairs the partial derivation into a boxed final answer that the verifier then promotes. On `mmlu-pro-test-08795`, the legacy/no-AMC numeric view misses a choice-form solution, while CACC+SP contributes an answer-mode-aligned candidate that recovers the final answer A. On `gpqa-diamond-train-recNFJjE5PPTqVJGv`, the wrongly selected

candidate carries scaffold/instruction residue, illustrating that hygiene (rather than further sampling) determines whether the verifier latches onto a correct option. Together these cases trace a one-to-one mapping from interface failure modes to the oracle / V|O / final movements reported above.

6 Discussion

Composability and deployment. Because the offline rulebook is frozen before online inference, the same rulebook composes with proposer-side strengthening or conversion-aware rescue without changing the verifier. For deployment, an interface audit is a low-cost first step before investing in further sampling or feedback rounds. The MMLU-Pro readout chain and GPQA Diamond results show that interface alignment and candidate repair can convert latent oracle coverage into final accuracy.

Limitations. The current study targets numeric and MCQ answer modes with benchmark-locked parsers. Extending to free-form, code-generation, and multi-turn tasks requires generalizing the answer-mode registry beyond fixed parser contracts. The offline rulebook is manually calibrated per benchmark, and learning the routing policy end-to-end from selector feedback would reduce setup cost. Additional experiments show that the effect of CACC depends on the selector, readout, and evaluation scope; the accuracy decline in the original MMLU-Pro setting illustrates this limitation. The current ablation covers CCR+AMC, CCR-only, and hygiene-only transitions (Tables 2–3). Finer parser-only ablations are left to future work.

7 Conclusion

This paper identified the hidden interface tax in multi-candidate generate-then-rerank reasoning, where correct candidates appear in the pool but cannot be consumed by the fixed selector. We formalized this as the coverage–conversion decomposition, introduced selectable coverage as the operative upper bound, and proposed CACC to close the gap through offline-calibrated fragment repair, answer-mode alignment, and candidate hygiene. On numeric reasoning, CACC raises both oracle and final accuracy under a fixed verifier. On GPQA Diamond, CACC alone yields $+13.6$ pp and composition with proposer-side strengthening reaches $+20.2$ pp, demonstrating that interface-level intervention can convert latent oracle coverage into final accuracy without retraining any component.

Acknowledgments and Disclosure of Funding

This research received support from the National Natural Science Foundation of China (grants 62272050 and 62302048); the Guangdong Provincial Department of Education's 2023–2024 support for the Guangdong Key Lab of AI and Multi-modal Data Processing at Beijing Normal-Hong Kong Baptist University (BNBU), Zhuhai; the Institute of Artificial Intelligence and Future Networks (BNU-Zhuhai); the Engineering Center of AI and Future Education of the Guangdong Provincial Department of Science and Technology, China; and the Zhuhai Science-Tech Innovation Bureau (grant 2320004002772). We also acknowledge computing support from the Interdisciplinary Intelligence SuperComputer Center of Beijing Normal University (Zhuhai).

Competing interests: The authors declare no competing interests.

References

[1] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

[2] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024. URL https://arxiv.org/abs/2408.03314.

[3] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[4] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

[5] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

[6] Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *arXiv preprint arXiv:2502.06703*, 2025. URL https://arxiv.org/abs/2502.06703.

[7] Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.09677.

[8] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

[9] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[10] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024. URL https://arxiv.org/abs/2406.06592.

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

[12] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[13] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[14] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

[15] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pages 10764–10799. PMLR, 2023.

[16] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

[17] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024. URL https://arxiv.org/abs/2403.09629.

[18] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V Le, Denny Zhou, Swaroop Mishra, Huaixiu S Zheng, et al. Self-discover: Large language models self-compose reasoning structures. *Advances in Neural Information Processing Systems*, 37:126032–126058, 2024.

[19] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023. URL https://arxiv.org/abs/2310.04406.

[20] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690, 2024.

[21] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

[22] Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*, 2024.

[23] Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv preprint arXiv:2411.14405*, 2024.

[24] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332, 2025.

[25] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2401.10020.

[26] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2312.04474.

[27] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17452.

[28] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024. URL https://arxiv.org/abs/2409.12917.

[29] Yifei Wang, Yuyang Wu, Zeming Wei, Stefanie Jegelka, and Yisen Wang. A theoretical understanding of self-correction through in-context alignment. *Advances in Neural Information Processing Systems*, 37:89869–89912, 2024.

[30] Yanyang Li, Michael R Lyu, and Liwei Wang. Learning to reason from feedback at test-time. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5241–5253, 2025.

[31] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[32] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

[33] Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, et al. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*, 2025. URL https://arxiv.org/abs/2506.04178.

[34] Kefan Song, Amir Moeini, Peng Wang, Lei Gong, Rohan Chandra, Shangtong Zhang, and Yanjun Qi. Reward is enough: Llms are in-context reinforcement learners. *arXiv preprint arXiv:2506.06303*, 2025.

[35] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[36] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[37] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022. URL https://arxiv.org/abs/2212.08073.

[38] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[39] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[40] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[41] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797, 2025.

[42] Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Rm-bench: Benchmarking reward models of language models with subtlety and style. *arXiv preprint arXiv:2410.16184*, 2024.

[43] Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*, 2023. URL https://arxiv.org/abs/2308.00436.

[44] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024. URL https://aclanthology.org/2024.acl-long.510/.

[45] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

[46] Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, 2025.

[47] Ammar Ahmed, Azal Ahmad Khan, Ayaan Ahmad, Sheng Di, Zirui Liu, and Ali Anwar. Retrieval-of-thought: Efficient reasoning via reusing thoughts. *arXiv preprint arXiv:2509.21743*, 2025. URL https://arxiv.org/abs/2509.21743.

[48] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[49] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[50] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

[51] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

[52] Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024. URL https://arxiv.org/abs/2406.19314.

[53] Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Dylan Slack, Qin Lyu, et al. A careful examination of large language model performance on grade school arithmetic. *arXiv preprint arXiv:2405.00332*, 2024. URL https://arxiv.org/abs/2405.00332.

[54] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

[55] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, 2024.

[56] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

[57] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

[58] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[59] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[60] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[61] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[62] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[63] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

[64] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[65] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[66] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[67] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[68] Team Qwen. Qwq: Reflect deeply on the boundaries of the unknown, november 2024, 2024. URL https://qwenlm.github.io/blog/qwq-32b-preview/.

[69] Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, et al. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *arXiv preprint arXiv:2504.09037*, 2025.

[70] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

A Experimental Protocol, Data Scope, and Reproducibility

The main experiments use a generate-then-rerank protocol. Aside from the candidate-construction layer, comparisons keep benchmark split, retained cap, parser, scoring endpoint, and downstream selector fixed. Numeric tasks use final numeric exact match, while MCQ tasks are hard-locked to `choice_letter`. Table 1 in the main text reports the main results, and Appendix Tables 2 and 3 report auxiliary decomposition and interaction evidence. This appendix consolidates the experimental protocol, data scope, statistical conventions, and reproducibility interfaces, ensuring that every reported number traces back to a run manifest, prediction record, or aggregate diagnostic summary.

The model stack is as follows. The main candidate generator is Qwen3-1.7B. The fixed selector in the main experiments is a Qwen3-1.7B model fine-tuned on GSM8K training-split candidate pairs with a binary yes/no objective (checkpoint id `qwen3_17b_gsm8k512_verifier_yesno_v1`). The verifier was not trained on MMLU-Pro, GPQA, or any evaluation split used in this paper. CACC+SP rows use a Qwen3-8B proposer / prompt-mixture variant for proposer-side strengthening. The FTTT

reference uses its official test-time feedback/training protocol and is therefore reported as a high-compute external reference. Table 5 summarizes split, answer mode, and paper use for each evidence unit.

Table 5: Evidence units, sizes, and core metrics. The table lists sample size, answer mode, oracle, final, and V|O for the main evidence units in the main text.

Evidence unit	N	Mode	Oracle	Final	V|O	Use in paper
GSM8K Base	1319	numeric	644/1319	379/1319	0.5885	Numeric baseline
GSM8K CACC	1319	numeric	843/1319	479/1319	0.5682	Numeric main result
CompMath Base	3199	numeric	1027/3199	486/3199	0.4732	Numeric baseline
CompMath CACC	3199	numeric	1419/3199	825/3199	0.5814	Numeric main result
MMLU no-AMC numeric	12032	numeric	444/12032	115/12032	0.2590	Archived collapse provenance
MMLU MCQ reparse	12032	`choice_letter`	4900/12032	2842/12032	0.5800	Readout recovery
MMLU CACC	12032	`choice_letter`	5736/12032	2740/12032	0.4777	MCQ Gap exposure
MMLU CACC+SP	12032	`choice_letter`	6650/12032	3532/12032	0.5311	Proposer-side narrowing
GPQA Base	198	`choice_letter`	38/198	11/198	0.2897	MCQ Baseline
GPQA CACC	198	`choice_letter`	90/198	38/198	0.4222	MCQ Compatibility result
GPQA CACC+SP	198	`choice_letter`	97/198	51/198	0.5258	Proposer-side narrowing
Power Sampling x CACC	1319	numeric	0.7195	0.3381	0.4700	Diagnostic widening case

Data and statistical records are saved together with the run manifest. Each manifest contains at least the dataset/split id, sample filtering rules, candidate source, retained cap, parser endpoint, scoring script, selector checkpoint, random seed, prediction JSONL path, and figure/table generation script, so that every number in the main text traces back to a prediction record or aggregate diagnostic summary. Paired closure and rescue rows can recompute McNemar tests and bootstrap confidence intervals, while Power Sampling x CACC is used as a summary-only diagnostic.

Table 6 summarizes the inference operations in the audited configuration. CACC adds one batched generation of four completion sequences per example and rescoring of the modified candidate pool. The SP rows use a Qwen3-8B proposer.

Table 6: Inference operations in the audited configuration.

Component	Work and accounting boundary
Upstream generation	Audited contract: 16 sequences, at most 128 new tokens each; pool construction precedes retention.
Completion construction	One additional batch of 4 sequences, at most 160 new tokens each, per example.
Routing / normalization	Deterministic diagnostics, parsing, sanitization, and deduplication.
Fixed selector	Two yes/no continuation scores per retained candidate; batching determines forward-pass count.
Stronger proposer	Qwen3-8B replaces or augments the 1.7B source in the composition setting; accounted separately.

Specifically, GSM8K clean / full, `competition_math:numeric`, MMLU-Pro test, GPQA Diamond, MMLU-Pro confirm-512, and GPQA-128 each record split source, sample size, filtering rules, parser endpoint, and run id; smoke evidence and full/confirm evidence are tagged separately to maintain traceability across evidence scales. Paired closure uses McNemar tests with bootstrap confidence intervals. Code, configurations, and reproduction instructions are available at https://github.com/drongzzz0/CACC-nips. This work studies the candidate-interface problem in small-model reasoning enhancement and does not involve personal data decisions or high-risk automated deployment; its main application boundary is low-cost, local, or intranet small-model reasoning systems.

B Coverage Decomposition and Parser Provenance

The auxiliary decomposition in Table 2 (main text) reports paired deltas across audit rows. All deltas are subtractions over the same set of frozen rows, with no extra experiments; positive final gains can come from oracle gain or V|O gain. The MMLU-Pro CACC row is the key mechanism point:

Table 7: GSM8K candidate-budget diagnostics: oracle, V|O, and final do not move synchronously when sample count or retained cap changes.

Diagnostic	Setting	Oracle	V|O	Final	ΔF	Note
Sample sweep	cap 8, samples 4	0.6422	0.5844	0.3753	ref.	Fewer samples but high conversion
Sample sweep	cap 8, samples 32	0.6429	0.4517	0.2904	-8.49pp	Similar oracle, lower conversion
Retained-cap sweep	cap 8	0.6391	0.5682	0.3632	ref.	CACC canonical retained cap
Retained-cap sweep	cap 16	0.8249	0.3998	0.3298	-3.34pp	Larger retained pool raises oracle but hurts conversion

Table 8: GSM8K same-pool verifier sensitivity (final-only).

Setting	Final	ΔF	Note
Canonical verifier	0.3632	ref.	CACC canonical selector
verifier128	0.3465	-1.67pp	Same-pool selector sensitivity check
verifier512	0.3594	-0.38pp	close to Canonical verifier final

even after oracle rises, answer-mode alignment and proposer-side strengthening are still required to convert it stably into final gain. Power Sampling x CACC is used as a coverage-pressure diagnostic, illustrating that raw coverage, V|O, and final must be tracked jointly.

MMLU-Pro parser, scoring, and reparse provenance. This subsection documents the parser and scoring provenance for the MMLU-Pro compatibility evidence. The `CCR-only` artifact corresponds to a legacy-mode candidate pool whose metadata remains numeric. Under the archived numeric exact-match scoring path, this legacy/no-AMC candidate-interface setting yields oracle `444/12032 = 0.0369` and verifier `115/12032 = 0.0096`. The benchmark-locked MCQ reparse reads the same legacy/no-AMC candidate-interface as oracle `4900/12032` and verifier `2842/12032`, supporting the AMC/readout chain in the main text. Table 9 summarizes this provenance.

Table 9: MMLU-Pro parser and scoring provenance: this comparison is a scoring/readout-path ablation, not an ablation between two independent candidate pools.

View	Parser / scoring path	N	Final	Oracle	V|O	Use
legacy/no-AMC	archived numeric exact match	12032	115/12032	444/12032	0.2590	Numeric scoring near-collapse
legacy/no-AMC	benchmark-locked MCQ reparse	12032	2842/12032	4900/12032	0.5800	Readout recovery
CCR+AMC	benchmark-locked MCQ reparse	12032	2731/12032	5745/12032	0.4754	Micro-ablation reference
CACC	benchmark-locked MCQ reparse	12032	2740/12032	5736/12032	0.4777	Gap exposure
CACC+SP	benchmark-locked MCQ reparse	12032	3532/12032	6650/12032	0.5311	Gap narrowing

The MCQ scoreboard in this paper uses benchmark-locked choice-letter reparsing. MMLU-Pro and GPQA are both hard-locked to the `choice_letter` answer mode. This comparison therefore tests whether the legacy/no-AMC candidate-interface becomes usable under the correct MCQ readout path, and is presented in the main text as a scoring/readout-path ablation.

C Recovery, Composition, and Interaction Evidence

The ablation/recovery/interaction matrix is reported in Table 3 (main text); this appendix expands the supporting details.

The main Second-family recovery evidence is in the recovery rows of Table 3 and Panel D of Figure 4; Figure 5 additionally visualizes the paired bootstrap intervals. Reproducing this analysis requires retaining paired example ids, before/after prediction JSONL, McNemar contingency counts, and bootstrap seed, so that the p-values and confidence intervals reported in the main text can be recomputed. The result supports recoverable second-family compatibility.

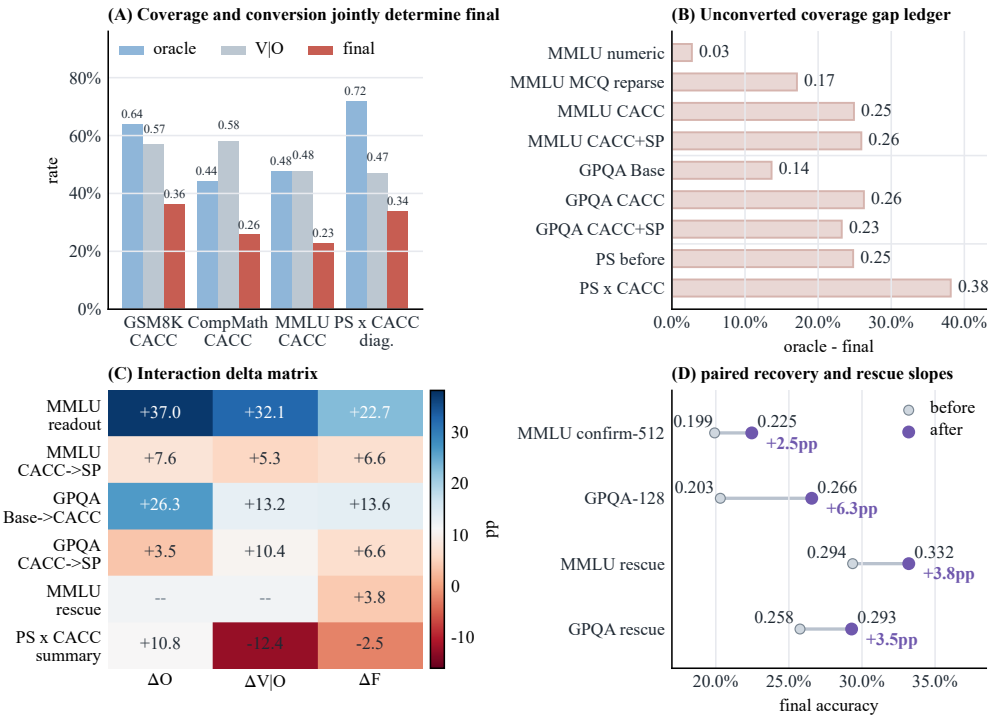

Figure 4: Mechanistic analysis of the coverage–conversion gap, compatibility, policy tradeoffs, and recovery closure. Panel A: oracle, V|O, and final decomposition. Panel B: unconverted gap between oracle and final. Panel C: interaction delta matrices for Δoracle, ΔV|O, and Δfinal. Panel D: before$\rightarrow$after slopes. Paired statistics are in Tables 3 and 10.

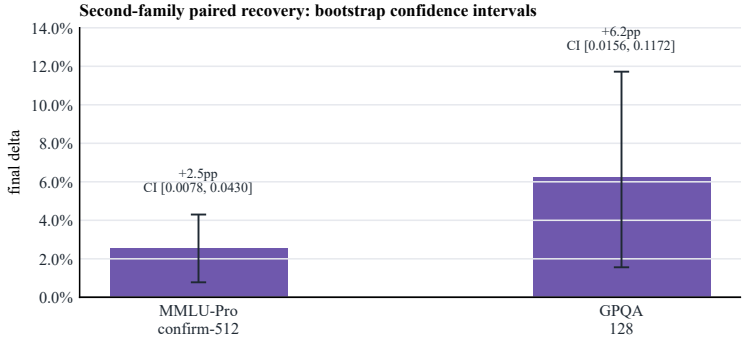

Figure 5: Bootstrap confidence intervals for second-family paired recovery. Both closure intervals lie above zero, supporting recoverable compatibility.

Layered composition and interaction data. This subsection documents the evidence scope of the layer-aware composition. Self-Refine corresponds to the trajectory-improvement layer, self-consistency to the candidate-source layer, ToT/search to the reasoning-space-expansion layer, FTTT to the feedback/test-time-training layer, and verifier/reranker to the selection layer. CACC corresponds to the candidate-interface layer. Table 3 reports the interaction evidence in the main text, and Table 11 further distinguishes frozen statistical results from summary-only diagnostics.

FTTT high-compute external reference. The archived FTTT GSM8K full log reports $1152/1319 = 0.8734$ with an average of 6.67 test-time trials. This reading corresponds to the official test-time feedback/training protocol; its sample budget, feedback loop, and model-state updates fall in a high-compute feedback/test-time-training regime. We use it as an external high-compute

Table 10: Paired statistics and diagnostic details for the mechanism rows in Table 3.

Evidence unit	Split / scope	Before	After	ΔF	w/l	Statistical detail	Artifact
Second-family recovery	MMLU confirm-512	102/512 (0.1992)	115/512 (0.2246)	+2.54pp	17/4	McNemar $p = 0.007197$; CI [0.0078, 0.0430]	Paired JSONL
Second-family recovery	GPQA-128	26/128 (0.2031)	34/128 (0.2656)	+6.25pp	9/1	McNemar $p = 0.02148$; CI [0.0156, 0.1172]	Paired JSONL
Selection rescue	MMLU-Pro	0.2936	0.3320	+3.84pp	613/150	McNemar $p = 3.43\times10^{-67}$	Paired pred.
Selection rescue	GPQA	0.2576	0.2929	+3.53pp	10/3	McNemar $p = 0.0923$	Paired pred.
Coverage pressure	GSM8K full	0.3632	0.3381	-2.50pp	summary	Oracle +10.84pp; V|O -12.43pp	Summary only
Second-family smoke	GSM8K-128	27/128 (0.2109)	29/128 (0.2266)	+1.56pp	7/1	Smoke row	Paired rows available
Second-family smoke	MMLU-128	24/128 (0.1875)	27/128 (0.2109)	+2.34pp	8/4	Smoke row	Paired rows

Table 11: Artifact availability matrix: artifact completeness and paper use of each evidence unit.

Evidence unit	Candidate pool	Prediction records	Paired statistics	Manuscript use
Main Base / CACC rows	available	available	row-level deltas	Frozen result rows
MMLU no-AMC numeric view	archived	available	provenance audit	Scoring/readout-path provenance
MMLU MCQ reparse	archived scoreboard	available	provenance audit	MCQ Readout recovery
Second-family closure	available	available	McNemar + bootstrap	Frozen paired recovery
Conversion-aware rescue	available	available	McNemar where available	Gap narrowing evidence
Power Sampling x CACC	summary + pool	summary metrics	aggregate diagnostic	Diagnostic summary-only

reference to illustrate the system-position difference between the candidate-interface layer and the feedback/test-time-training layer.

Table 12: External baseline provenance and artifact status.

Method	Task	Correct / N	Final	Extra cost	Artifact status
Self-Refine	GSM8K full	643/1319	0.4875	5.0 calls/example	Full row available
Self-Refine	CompMath numeric	789/3199	0.2466	17.0 calls/example	Full row available
Self-Refine	MMLU-Pro	2838/12032	0.2359	17.0 calls/example	Full row available
Self-Refine	GPQA Diamond	24/198	0.1212	17.0 calls/example	Full row available
Self-Refine	GSM8K-128	51/128	0.3984	eval-128	Small reference row
ToT-lite	GSM8K-128	37/128	0.2891	eval-128	Small search baseline row
CCQA	GSM8K-128	21/128	0.1641	eval-128	Small query-generation row
FTTT	GSM8K full	1152/1319	0.8734	6.67 trials/example	Final-only high-compute reference
PairRM	CompMath numeric	257/3199	0.0803	best-of-16	Primary report
PairRM	CompMath numeric	244/3199	0.0763	best-of-16	Secondary report; provenance discrepancy noted
PairRM	MMLU-Pro	793/12032	0.0659	best-of-8	Primary report

D Offline Rulebook Components

The policy-parameterized pseudocode of CACC is given as Algorithm 1 in Section 4. The offline rulebook components instantiated in our implementation are summarized below. The replacement policy is treated as a swappable component, while the method composition of CACC remains CCR+AMC+SCH. This is consistent with the offline–online flow in Figure 2: offline calibration produces a frozen rulebook, and online inference only executes gold-free routing and candidate construction.

Table 13: Offline rulebook components used by CACC.

Component	Diagnostic trigger	Online action	Frozen output
CCR	partial derivation, missing final answer, fragmented numeric form	repair final-answer span and recover parseable candidate text	candidate text plus parser status
AMC	Answer-mode mismatch, missing choice letter, Wrapper drift	map candidate to benchmark-locked mode and protect choice-letter wrapper	mode-aligned candidate
SCH	duplicated scaffold, template residue, selector-attractive noise	remove residue, deduplicate, and keep retained cap stable	hygienic retained pool
Routing policy	diagnostic vector ϕ and retained cap K	select rule sequence without gold access	frozen $\pi(\phi, K)$

E Interface Error Taxonomy and Rule Interfaces

Table 14 lists the candidate-interface error classes used in our qualitative analysis.

Table 14: Candidate-interface error taxonomy used in the qualitative analysis.

Error class	Diagnostic signal	Expected effect
Oracle miss	No candidate parses to gold	Low oracle
Fragment / partial	Reasoning stops before final answer	Low oracle / Low V|O
Answer-mode mismatch	Numeric answer in MCQ or missing choice letter	Low MCQ final
Invalid final	Malformed final channel	Parser failure
Scaffold residue	Instruction or template leakage	Selector confusion
Wrapper drift	Second-family answer wrapper differs	Transfer loss
Oracle-hit verifier miss	Correct candidate present but not selected	Low V|O

This taxonomy makes explicit the difference between raw candidate presence and selectable coverage. Each error class corresponds to one path of candidate-interface failure, showing how a candidate can affect the final decision through Parser failure, Answer-mode mismatch, or Selector confusion.

Prompts, templates, parsers, and sanitizers. This subsection records the reproducibility interfaces of the CCR prompt template, AMC answer wrapper, MCQ choice-letter lock, numeric parser normalization, sanitizer rules, final answer extraction, and benchmark-specific prompt notes. All prompt and parser entries are bound to the run manifest, simplifying reproduction of the experimental candidate construction and scoring path.

Evidence scope and claim mapping. Table 17 maps the main claims (numeric final gain, MCQ answer-mode alignment, Second-family recovery, proposer-side strengthening, selection-side rescue, and coverage-pressure diagnostic) to the evidence blocks of Tables 1, 2, and 3.

Table 15: Representative audit examples. These examples are used for post-hoc qualitative analysis only, not for offline rule calibration.

Example id	Dataset	Gold	Audited failure / recovery pattern
`gsm8k-test-00595`	GSM8K	13	Oracle miss; CACC verifier selects 15 although gold is 13
`competition-math-numeric-test-02878`	CompMath	13	raw oracle hit exists, but verifier overrules first candidate and selects 11
`mmlu-pro-test-08795`	MMLU-Pro	A	no-AMC numeric view misses a choice-form solution; MCQ-aware CACC+SP later recovers final answer A
`gsm8k-test-00462`	GSM8K	71	CACC repairs Fragment / partial candidate into a boxed final answer
`gpqa-diamond-train-recNFJjE5PPTqVJGv`	GPQA	B	selected wrong candidate contains scaffold / instruction residue, illustrating hygiene pressure
`mmlu-pro-test-08026`	MMLU-Pro	D	stronger proposer adds/selects D after CCR/no-AMC misses the answer
`mmlu-pro-test-00942`	MMLU-Pro	F	second-family CACC selects long option text; hygiene recovers bare choice letter F

Table 16: Prompt, parser, and sanitizer interfaces bound to run manifests.

Interface item	Constraint	Reproducibility role
CCR prompt template	no gold access; repair only observable fragments	reproduces Fragment / partial candidate repair
AMC answer wrapper	numeric tasks keep exact numeric form; MCQ tasks hard-lock to `choice_letter`	fixes answer-mode alignment and readout path
Numeric parser normalization	normalize whitespace, delimiters, final numeric span, and wrapper variants	keeps open-ended scoring stable across candidate sources
MCQ parser configuration	preserve benchmark option set and final choice-letter endpoint	supports MMLU-Pro / GPQA reparse provenance
SCH sanitizer rules	remove Scaffold residue, duplicated wrappers, and selector-attractive debris	stabilizes verifier input distribution
Manifest binding	every prompt/parser/sanitizer version is tied to run id, seed, split, and scorer	enables rerunning table and figure generation scripts

Table 17: Evidence scope and claim mapping.

Claim	Evidence unit	Paper use
CACC improves numeric open-ended reasoning	GSM8K + competition_math	Direct final-gain evidence
MCQ benefits from AMC / scoring-path alignment	MMLU-Pro readout chain	Answer-mode alignment evidence
Second-family compatibility is recoverable	MMLU confirm-512 / GPQA-128	Paired recovery evidence
CACC+SP narrows MCQ gap	MMLU-Pro / GPQA	Proposer-side composition evidence
Conversion-aware rescue improves final selection	MMLU-Pro / GPQA	Selection-side composition evidence
Power Sampling x CACC diagnoses Coverage pressure	aggregate summary	Coverage–conversion diagnostic

This mapping separates frozen results, paired recovery, and diagnostic summaries by their paper roles, ensuring a one-to-one correspondence between main-text claims and auditable artifacts.

F Notation Summary and Per-Benchmark Setup

Table 18 consolidates the symbols used across the paper, and Table 19 summarizes per-benchmark setup details (split, parser, scoring endpoint, sample budget) so that each row in the main results can be reproduced from the manuscript without prior context.

Table 18: Notation summary. Symbols are introduced in Sections 3 and 4 and used throughout the paper.

Symbol	Meaning
$x, y_x, \mathcal{D}$	input instance, ground-truth answer, evaluation dataset
$m \in \{\text{numeric}, \texttt{choice_letter}\}$	benchmark-locked answer mode, fixed by registry $\mathcal{M}$
$C_0(x) = \{c_1, \ldots, c_n\}$	raw candidate pool of size n produced by the generator
$T_m, R_K, C_K(x, m)$	mode-aware candidate construction, retained-cap controller, retained pool of size $\leq K$
$\phi(x, c, m)$	gold-free diagnostic feature vector (parseability, Answer-mode mismatch, residue, ...)
$\mathcal{R}, \pi$	frozen rulebook (CCR/AMC/SCH) and routing policy $\pi(\phi, K) \mapsto a$
$S, \hat{c}, \hat{y}$	fixed selector (verifier/reranker), selected candidate, parsed final answer
$A_{\text{first}}, A_{\text{oracle}}, A_{\text{verifier}\mid\text{oracle}}, A_{\text{final}}$	first-answer accuracy, oracle coverage, conditional V|O, final accuracy
$A^{\text{sel}}_{\text{oracle}}, H_{\text{sel}}$	selectable oracle coverage and the selectable-correct hit event (Eq. 8)
CACC, CCR, AMC, SCH	method, completion-centric replacement, answer-mode compatibility, selector-compatible hygiene
SP	proposer-side strengthening (Qwen3-8B prompt-mixture proposer) used in composition rows

Table 19: Per-benchmark setup details. The retained cap is shared across compared methods on each benchmark.

Benchmark	Split / N	Mode	Retained cap	Parser / scoring endpoint
GSM8K	clean test, N=1319	numeric	8	numeric exact match on final boxed span
`competition_math:numeric`	numeric subset, N=3199	numeric	8	numeric exact match on normalized final span
MMLU-Pro	test, N=12032	MCQ	8	benchmark-locked choice-letter reparse
GPQA Diamond	full, N=198	MCQ	8	benchmark-locked choice-letter reparse
MMLU-Pro confirm-512	confirm split, N=512	MCQ	8	second-family paired closure run
GPQA-128	subset, N=128	MCQ	8	second-family paired closure run
GSM8K-128 / MMLU-128	smoke subsets, N=128	numeric/MCQ	8	Mistral-7B Second-family smoke runs

The two tables together with the run-ledger in Table 5 make every row in Tables 1–3 traceable to its split, parser, scoring endpoint, sample budget, and answer mode.

G Controlled Evaluation Protocols and Detailed Results

G.1 Experimental units and reporting conventions

We identify each result by candidate pool, selector, readout, and evaluation set. Original-pool controls reuse the candidate contents underlying the historical results; fresh-generation controls create new pools and have separate baselines. Accuracies are measured per question. Three-seed verifier means are descriptive summaries, with per-seed paired intervals reported below. We do not treat repeated seeds on the same questions as independent question samples.

G.2 Dataset-specific verifier supervision

Verifiers use Qwen3-1.7B with response-only yes/no loss. MMLU-Pro supervision is restricted to the official validation set (70 questions, split into 56 train and 14 dev before format expansion). Evaluation on the 11,520-question held-out subset excludes the inspected confirm-512 subset. GPQA

supervision comes from Extended after excluding Diamond overlaps using record identifiers and normalized question–option hashes; the cleaned set has 346 questions (278 train, 68 dev). Canonical and format-diverse option renderings are balanced 1:1 between yes and no within each format and split. Training uses three epochs, learning rate 2×10^{-4}, LoRA rank 8, alpha 16, dropout 0.05, batch size 1, gradient accumulation 4, and maximum length 768. Prompt/padding labels are masked; truncation is forbidden. Checkpoints are selected on grouped dev accuracy, then dev loss and binary accuracy, without evaluation labels. The fixed seeds are 14251, 20260724, and 314159.

Raw readout scores candidate text. Canonical readout maps a supported option to a common `Final answer: <LETTER>` representation and reuses its score across candidates supporting that option. It neither regenerates candidate reasoning nor adds a previously unsupported answer option. Within each target and seed, Base and CACC use the same verifier. The GPQA canonical means are 16.16% and 24.92%; MMLU held-out means are 26.40% and 27.84%. Mean raw-readout gains are 9.93 pp and 0.37 pp, respectively, illustrating the effect of the readout contract.

Table 20: Target-verifier canonical results on original pools. CIs are paired per-seed 95% bootstrap intervals for ΔF (pp); p is the per-contrast exact McNemar value, without a cross-row multiplicity adjustment.

Evaluation	N	Seed	Base	CACC	ΔF	95% CI	p
GPQA	198	14251	15.66	24.24	+8.59	[2.53,15.15]	.0137
GPQA	198	20260724	16.67	26.26	+9.60	[3.54,15.66]	.00432
GPQA	198	314159	16.16	24.24	+8.08	[1.52,14.65]	.0259
MMLU held-out	11,520	14251	26.35	28.07	+1.73	[1.04,2.43]	6.48×10^{-7}
MMLU held-out	11,520	20260724	26.52	28.15	+1.63	[0.94,2.34]	4.23×10^{-6}
MMLU held-out	11,520	314159	26.35	27.31	+0.96	[0.26,1.68]	.00799

A smaller validation-only control trains three format-matched response-only verifiers on the same 56/14 MMLU split. On frozen GPQA pools its gains are $+7.07$, $+11.11$, and $+8.08$ pp; on MMLU confirm-512 they are $+2.34$, $+1.17$, and $+0.98$ pp, with the latter per-seed intervals crossing zero. This control evaluates small-scale format-matched MCQ supervision separately from the dataset-specific canonical-readout study.

G.3 Paired constrained decoding

The four-cell experiment uses Qwen3-1.7B generation and the target-specific canonical verifier at seed 14251. Base trajectories and CD prefixes are paired token-for-token; independently sampled CD trajectories are not substituted. Supported canonical letters are scored once per question and shared across all four cells. The primary contrast family contains CACC minus Base, CD minus Base, and CD+CACC minus CD, with exact McNemar tests and Holm adjustment. Confidence intervals use 10,000 paired question resamples with seed 14251.

Table 21: Fresh-generation four-cell outcomes. O, $V|O$, and F are percentages.

Evaluation	Cell	O	$V\|O$	F
GPQA ($N = 198$)	Base	20.71	82.93	17.17
	CACC	51.01	45.54	23.23
	CD	75.25	39.60	29.80
	CD+CACC	81.82	35.19	28.79
MMLU confirm ($N = 512$)	Base	31.45	72.05	22.66
	CACC	48.83	60.00	29.30
	CD	62.30	48.59	30.27
	CD+CACC	68.16	41.26	28.12

The Holm-adjusted primary p-values, in contrast order, are .1923, .00668, .7539 on GPQA and .001764, .002182, .02660 on MMLU confirm-512. The interactions on F are -7.07 pp (CI $[-14.14, 0.00]$) and -8.79 pp (CI $[-12.70, -4.88]$). The additional direct CD-minus-CACC contrasts are $+6.57$ pp on GPQA (unadjusted $p = .04096$, CI $[+1.01, +12.63]$) and $+0.98$ pp on

MMLU (unadjusted $p = .6718$, CI $[-2.73, +4.50]$). These additional contrasts are outside the primary Holm family.

G.4 Selector families

Majority voting and the three LLM scoring variants operate on the original candidate pools. Option-logprob scoring uses an 8B model; candidate judges use 1.7B and 32B models, with the latter Qwen3-32B. The strong judge's candidate-text limit is 2,000 characters and no evaluated candidate is clipped. The MMLU majority row uses the full 12,032 questions, while the model-based MMLU rows use confirm-512; their sample scopes are shown explicitly.

Table 22: Selector-family results on original pools. Accuracies are percentages; p is the exact paired McNemar value for each row, without a table-wide adjustment.

Evaluation	N	Selector	Base	CACC	ΔF	p
GPQA	198	Majority	16.67	19.70	+3.03	.2101
GPQA	198	8B option-logprob	17.68	28.28	+10.61	.00460
GPQA	198	1.7B candidate judge	5.56	18.69	+13.13	4.23×10^{-5}
GPQA	198	32B candidate judge	6.57	23.23	+16.67	1.07×10^{-6}
MMLU full	12,032	Majority	21.17	18.48	-2.69	3.43×10^{-18}
MMLU confirm	512	8B option-logprob	32.62	32.62	0.00	1.000
MMLU confirm	512	1.7B candidate judge	18.75	17.97	-0.78	.7709
MMLU confirm	512	32B candidate judge	23.83	25.20	+1.37	.5825

The strong-judge paired intervals are $[+10.10, +23.23]$ pp on GPQA and $[-2.73, +5.47]$ pp on MMLU confirm-512. Candidate coverage is identical across selector families within each pool and evaluation scope; final accuracy changes with selection.

G.5 Five-seed generator study

Mistral-7B-Instruct-v0.1 generates fresh pools for MMLU confirm-512 at seeds 7, 11, 19, 23, and 31, temperature 0.7, and top-p 0.95. Base generation samples 16 sequences with at most 128 new tokens each. CACC adds four completion sequences with at most 160 new tokens each, and both conditions retain at most eight candidates. The selector is the Qwen3-1.7B MMLU adapter without Mistral-specific retraining. Mean oracle coverage rises from 48.20% to 58.36%, and mean final accuracy from 14.22% to 15.94%. All five final deltas are positive. Paired-question bootstrap of the fixed-five-seed mean gives $+1.719$ pp with a 95% interval $[+0.742, +2.773]$ pp. This study varies the generator while keeping the selector and retained cap fixed; completion generation remains an additional operation.

G.6 Format-only intervention and pool diagnostics

The paired format intervention changes only the final-answer marker, between `Final answer:` and `Option <LETTER>`; reasoning text and decoded answer content are fixed. Three format-matched validation-trained verifiers are evaluated on MMLU confirm-512. For the CACC condition, 381 of 512 questions meet the prespecified eligibility rule. Fixed-three-seed effects are averaged within questions before 10,000 question-level bootstrap resamples. The signed margin shift (Final answer minus Option) is -0.3899 (95% CI $[-0.5881, -0.1940]$; exact two-sided sign-test $p = .0027$). Eligible selection flips are 9.89% (CI $[7.96, 11.90]\%$). All-question accuracy changes by -0.065 pp (CI $[-0.911, +0.716]$ pp). The estimand is format sensitivity of selection, not an aggregate accuracy gain.

For the full MMLU majority path, reconstructed snapshot differences identify 1,620 newly covered examples, 4,116 jointly covered examples, and 784 examples losing correct coverage. Of the new hits, 261 convert (16.11%). Conversion on jointly covered examples changes from 57.31% to 47.67%. These comparisons describe the two recorded candidate pools rather than a traced generation lineage.

G.7 Prospective ARC-Challenge validation

We froze the schema wrapper, operative MCQ rules, model, parser, sampling settings, and evaluation protocol before a single paired evaluation on 256 ARC-Challenge test questions [70]. Questions were selected by sorting SHA256(`CACC-ARC-v1|source_id`) without labels. Original option labels were mapped in their existing order to A–J. The existing operative payload required zero edits; only the schema/label wrapper was added. Gold labels were excluded from model inputs and accessed only after both predictions were frozen.

Base uses Qwen3-1.7B, 16 samples of at most 128 new tokens, temperature 0.7 and top-p 0.95, retaining at most eight candidates. CACC adds four completions of at most 160 new tokens using the existing benchmark-aware prompt, three context attempts, one protected prefix candidate, and fragment-first replacement. The frozen format-matched MMLU validation-trained verifier (seed 14251, epoch 1) scores candidate-supported canonical letters with one shared score cache. Sampling uses seed 14251, with deterministic per-question offsets to support checkpointed execution. This single-seed study measures reuse on a new benchmark under a fixed selector.

Table 23: Prospective ARC-Challenge results ($N = 256$). Oracle (O), conditional conversion (V), and final accuracy (F) are percentages.

Condition	O	V	F
Base	85.16	82.11	69.92
CACC	91.02	77.68	70.70

Final accuracy changes by $+0.78$ pp (10,000 paired-question bootstrap resamples; 95% CI $[-2.34, +3.91]$ pp; exact two-sided McNemar $p = 0.8036$). The final-accuracy difference is not statistically significant.

The cumulative single-GPU session ledger records 0.548 GPU allocation wall-clock hours, including runtime preparation probes and a conservative charge for a timed-out probe; this is not a measurement of GPU busy time or energy. The recorded implementation-and-runtime setup window spans 19.1 minutes from the first saved preflight script to protocol freeze, including compatibility checks and orchestration. This is an agent workflow wall-clock interval; human adaptation time was not measured.

G.8 Release provenance and adaptation measurement

The public implementation is maintained at `https://github.com/drongzzz0/CACC-nips`. The inspected release version is commit `8fdf207`. Its per-row ledger distinguishes original artifacts, reproduced display ratios, reconstructed reruns, and missing prediction-level provenance. Some stronger-proposer rows have reconstructed reruns rather than the exact original candidate streams; these are documented as separate protocols. The original MMLU Base pair is unavailable for a new paired test of the historical -0.85 pp contrast. New controls above use their own paired records and do not substitute for that pair.

A source audit finds a common operative MCQ payload across GPQA and MMLU-Pro, with benchmark-specific wrappers. This is retrospective implementation reuse. Appendix G.7 adds a prospective study with the benchmark, rulebook, model, parser, and metric frozen before evaluation; agent setup time and compute are recorded separately from unmeasured human effort.

Broader impact. Improving smaller models can support local and resource-constrained reasoning applications. Better answer formatting also makes an incorrect answer easier to consume, so parser validity should be monitored together with correctness. The framework does not provide correctness guarantees or replace application-specific validation. Release materials should respect benchmark licenses and access restrictions, including the handling of GPQA content.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the candidate-interface problem, the coverage–conversion formulation, and the CACC construction mechanism. They distinguish numeric reasoning gains from MCQ coverage–conversion failures and separate CACC from proposer-side strengthening. Sections 3–5 and Section 6 bound the selector, task, and composition scope rather than claiming universal accuracy gains or equal total inference cost.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the restricted answer modes, benchmark-specific rule calibration, selector/readout dependence, and the limits of the ablations. Appendix G reports negative or inconclusive controls, separates evaluation scopes, and distinguishes retrospective reuse from the prospective ARC study. Its release-provenance discussion identifies missing original prediction-level records and reconstructed reruns.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theorems. Section 3 defines the coverage quantities and states the extractive-selector condition for the oracle–conversion identity. Equation 10 follows from the law of total probability, with zero-probability branches and the residual term explicitly handled; Equation 11 states the resulting bound and its applicability conditions.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 4–5 and Appendices A, D, and G describe the construction procedure, model/checkpoint identities, benchmark splits, parser contracts, retained caps, and audited generation operations. Tables 10, 11, and 16 identify paired evidence, availability boundaries, and prompt/parser interfaces. These support protocol-level reproduction; the release-provenance discussion in Appendix G explicitly distinguishes reconstructed and summary-only evidence from original records and does not claim exact replay of every historical row.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Code, configurations, and reproduction instructions are available at `https://github.com/drongzzz0/CACC-nips`, as described in Appendices A and G. The source-only release excludes benchmark datasets, model weights/adapters, raw candidate pools, and prediction logs. Some original candidate streams are unavailable and some rows have reconstructed reruns rather than exact original provenance; we therefore do not claim a complete open reproduction bundle for all reported results.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

Justification: Section 5 and Appendix A identify the main generator, fixed selector, proposer variant, splits, scoring endpoints, and retained caps. Appendix G gives the additional-control supervision splits, LoRA settings, learning rate, batch/accumulation settings, epochs, checkpoint selection, sampling parameters, and seeds. Comparisons distinguish fixed-cap matching from total computation: completion generation and rescoring are additional operations, not equal-cost inference.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix Table 10 reports paired recovery/rescue statistics. Appendix G specifies paired-question bootstrap intervals, exact McNemar or sign tests, the Holm-adjusted primary constrained-decoding family, and the unit of resampling for repeated-seed summaries. Per-seed descriptive results are not treated as independent question samples; additional contrasts and summary-only diagnostic evidence are identified separately, and the ARC final-accuracy difference is reported as not statistically significant.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Table 6 discloses the audited generation, additional completion, and rescoring operations, while Appendix G.7 reports the prospective study's GPU-allocation accounting. However, the manuscript does not provide a complete worker/memory/execution-time ledger for every historical experiment. The fixed retained cap is not an equal-compute claim, and the cost of completion generation is not assumed negligible.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics?

   Answer: [Yes]

   Justification: The work evaluates candidate construction, model inference, and verifier training on existing reasoning benchmarks, without human-subject experiments, personal-data collection, or high-risk automated deployment. Appendix G discusses the application boundary, incorrect-answer risks, and benchmark access restrictions. The candidate-construction policy uses no test-time gold labels, and the prospective ARC study separates protocol/prediction freezing from gold-label evaluation.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The broader-impact paragraph at the end of Appendix G discusses support for local and resource-constrained reasoning, while noting that improved answer formatting can also make incorrect answers easier to consume. It calls for monitoring correctness alongside parser validity and does not present the framework as a correctness guarantee or a replacement for application-specific validation.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse?

    Answer: [N/A]

    Justification: The released artifact is method code and configurations, not a new foundation model, a scraped dataset, or an unrestricted release of benchmark content. The release excludes benchmark data and model weights. The broader-impact discussion in Appendix G calls for respecting benchmark licenses and access restrictions, including GPQA.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The paper credits benchmark and model creators through their original papers and notes the need to respect licenses and access restrictions in Appendix G. However, it does not provide a complete asset-by-asset account of the exact licenses and terms of use for all third-party data, models, and code. We therefore do not mark the full explicit-license-disclosure requirement as satisfied.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The method's code release provides a README with setup instructions, synthetic smoke-test examples, pipeline entrypoints, and configuration guidance, together with release-inventory and reproduction-status documentation. The repository is linked in Appendices A and G; its documentation identifies excluded artifacts and reproduction caveats. No new benchmark dataset or foundation-model checkpoint is introduced.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not include crowdsourcing experiments or human-subject research; its evaluations use model inference, verifier training, and automated scoring on existing benchmarks.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether IRB approvals (or an equivalent approval/review) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human subjects, so IRB or equivalent human-subject review is not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research?

    Answer: [Yes]

    Justification: The model stack in Section 5 and Appendix A identifies the Qwen3-1.7B generator and fixed yes/no verifier, and the Qwen3-8B proposer in the SP setting. Appendix G describes the trained control verifiers, alternative selectors, and Mistral generator study. The CACC construction layer is distinguished from these separately trained or substituted model components.