Constraint-First Reasoning: A Training-Free Protocol for Exploiting Answer-Space Constraints in Mathematical Problem Solving

Anonymous ACL submission

Abstract

Large language models solve competition-level mathematics with high accuracy, yet frequently violate explicit answer-space constraints—returning non-integer values when integers are required, ignoring modular reductions, or producing answers outside specified ranges. We introduce **Constraint-First Reasoning** (CFR), a training-free two-stage inference protocol that extracts and propagates answer-space constraints *before* solving, then guides the solver to reason within the feasible set. Stage 1 identifies restrictive lexical cues and cross-propagates them to narrow the answer range. Stage 2 solves under these explicit constraints with interleaved consistency checks. We further propose ROUTED-CFR, which prepends a regex-based router that activates the full pipeline only when constraint cues appear. In the main AIME/CMIMC evaluation in Table 1, CFR improves the reported backbone averages by $+3.3\%$ to $+8.8\%$ over direct CoT. In the DeepSeek-V4-Pro ablation in Table 2, Routed CFR improves BRUMO25 and CMIMC25, preserves AIMO_AMC accuracy, and uses lower token multipliers than CFR without routing. Table 10 further reports BRUMO25 and AIMO_AMC results for all four backbones. These results position constraint-first inference as a targeted test-time strategy with clear applicability boundaries.

1 Introduction

Competition mathematics problems embed constraints in natural language: "find the remainder when N is divided by 1000," "how many positive integers satisfy...," "express your answer as $p + q$ where p and q are coprime." These constraints define the *feasible answer region*—a small, structured subset of all possible values. A solver that ignores them risks producing answers that violate stated requirements despite correct underlying reasoning.

Why do constraint violations occur so frequently? We identify three root causes. First, *attention dilution*: in long reasoning chains, early constraint information can fade before the model reaches a final answer (Liu et al., 2024). Second, *implicit encoding*: models trained on solution traces handle constraints implicitly—they may notice a modular condition mid-solution or check integer constraints post-hoc—but this implicit handling breaks under distribution shift. Third, *format ambiguity*: problems with encoded answer formats ("find $m+n$ where $\gcd(m, n) = 1$") require multi-step post-processing that models frequently skip or misapply.

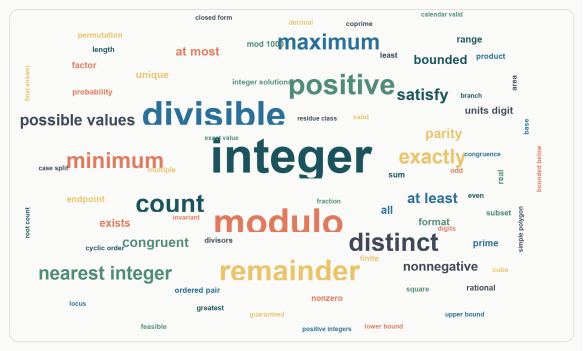

Figure 1: **Restrictive lexical cues in competition mathematics.** Larger words correspond to cue types emphasized by our router, including integer-answer formats, divisibility and modular conditions, bounds, extremal operators, exactness requirements, and counting restrictions. These cues motivate CFR: before solving, identify what any valid answer must satisfy.

Current reasoning models, trained to generate extended chains of thought (DeepSeek-AI, 2025; OpenAI, 2024), handle constraints implicitly. A model may follow a plausible derivation while leaving answer restrictions implicit until the end; explicit constraint tracking is designed to keep those restrictions visible throughout solving.

We propose **Constraint-First Reasoning** (CFR), a training-free protocol that makes constraint handling explicit and systematic. The

key insight: *extract constraints before solving, not after*. This mirrors human competition practice, where experienced contestants identify the required answer form before attempting a solution (Pólya, 1945). Front-loading constraint identification ensures the solver maintains awareness of answer-space requirements throughout its reasoning chain.

CFR operates in two stages:

1. **Constraint Extraction & Propagation**: Extract all answer-space constraints (domain, modular, parity, bounds, format) from the problem statement and cross-propagate them to derive the tightest feasible set.
2. **Constraint-Guided Solving**: Solve under the extracted constraints, checking intermediate results against the constraint specification at each major step.

We further introduce ROUTED-CFR, which prepends a regex-based router that checks whether the problem text contains restrictive lexical cues. Problems without such cues bypass the pipeline and proceed directly to standard chain-of-thought reasoning. This purely syntactic router decides whether to use the full pipeline before generation begins.

Our experiments use Table 1, Table 2, and Table 10 as the sources of empirical data. They reveal a clear and interpretable pattern:

- In Table 1, CFR improves the reported AIME/CMIMC averages for all four CFR backbones, with gains from $+3.3\%$ to $+8.8\%$.
- The largest average gains are reported for DeepSeek-V4-Pro ($+8.8\%$) and Qwen3.5-9B ($+8.5\%$), while Qwen3.5-35B-A3B and JustRL-1.5B show smaller average gains.
- In Table 2, Routed CFR is more accurate and uses lower token multipliers than CFR without routing on each reported DeepSeek-V4-Pro ablation benchmark.
- Table 10 extends the BRUMO25 and AIMO_AMC comparison to all four reported backbones.

These findings position CFR not as a universal reasoning enhancer but as a targeted intervention for settings where answer-space constraints are informative and the backbone can use them.

2 Related Work

2.1 Self-Refinement and Verification

Self-Refine (Madaan et al., 2023) iteratively improves outputs via self-feedback. Process reward models (Lightman et al., 2024; Wang et al., 2024) score individual reasoning steps to guide search. Outcome reward models (Cobbe et al., 2021) provide binary signals on final answers. ReAct (Yao et al., 2023b) interleaves reasoning with actions for grounded decision-making. Reflexion (Shinn et al., 2023) leverages verbal self-reflection to improve subsequent attempts. All these approaches verify or refine *after* generation. CFR instead supplies constraints *before* solving, enabling the solver to reason within the feasible set from the outset rather than detecting and correcting violations post-hoc.

2.2 Test-Time Compute Scaling

Recent work demonstrates that allocating more inference-time computation improves performance across diverse tasks. Best-of-n sampling (Brown et al., 2025) and compute-optimal scaling (Snell et al., 2025) establish scaling laws for test-time compute. S1 (Muennighoff et al., 2025) shows that budget forcing (extending reasoning with "wait" tokens) improves accuracy. Unlike generic compute scaling, CFR concentrates additional computation on constraint identification—a narrow task with compact output that produces high-value information guiding the larger solving budget.

3 Method

3.1 Problem Formulation

Let q denote a mathematical problem, $\mathcal{A}$ the answer space, y a reasoning trace produced by the model, and $\pi(y)$ the function that extracts a final answer from y. Standard inference samples:

$$a^* = \pi(y^*), \quad y^* = \arg\max_{y} P_\theta(y \mid q) \tag{1}$$

where P_θ is the language model. Many competition problems define a feasible answer region $\Phi(q) \subset \mathcal{A}$ through explicit constraints (e.g., "integer in 000–999" implies $|\Phi(q)| = 1000$ vs. $|\mathcal{A}| = \infty$). CFR makes $\Phi(q)$ explicit:

$$a^*_{\text{CFR}} = \arg\max_{a \in \Phi(q)} P_\theta(a \mid q, \Phi(q)) \tag{2}$$

The information-theoretic value of constraint extraction is:

$$I(A; C \mid Q) = H(A \mid Q) - H(A \mid Q, C) \tag{3}$$

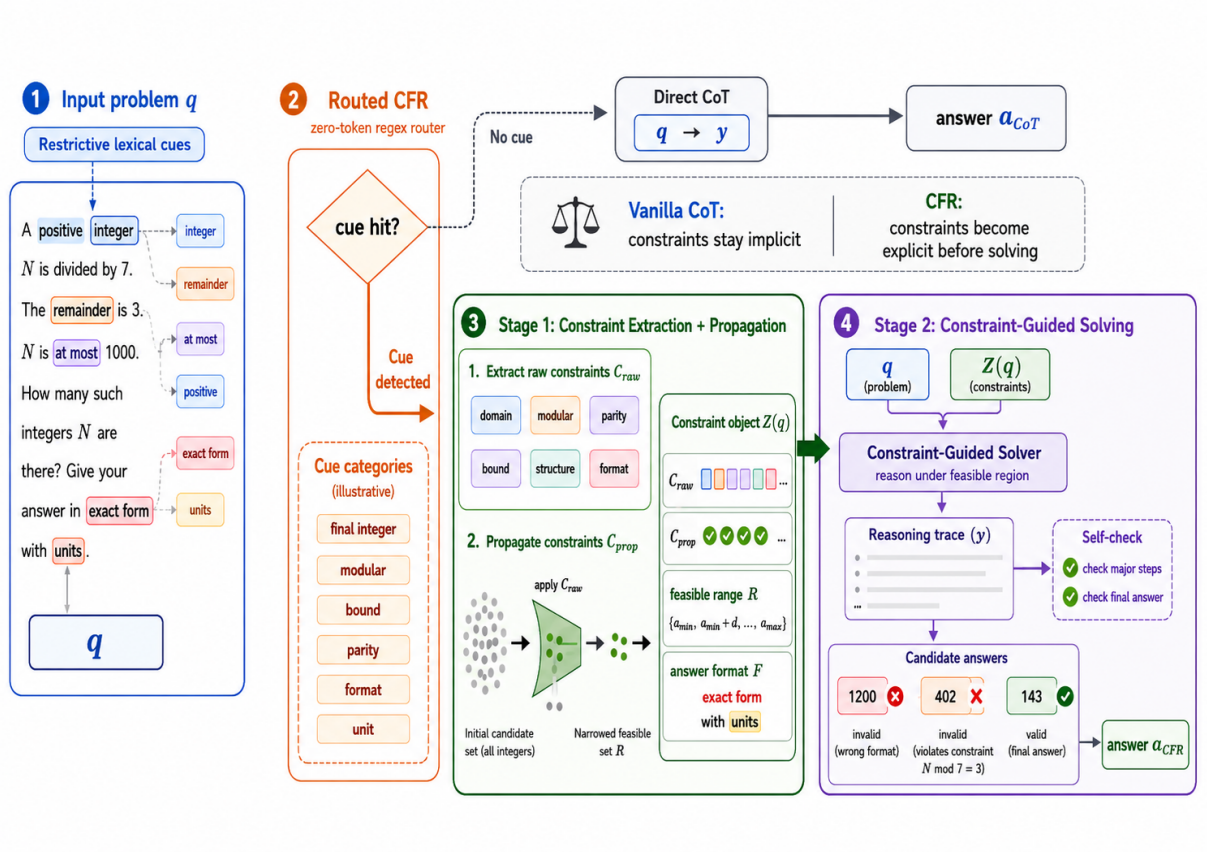

Figure 2: **ROUTED-CFR framework overview.** Given an input problem, the regex router checks for restrictive lexical cues. If detected, Stage 1 extracts and cross-propagates answer-space constraints (domain, modular, parity, bound, format), then Stage 2 solves under the resulting feasible set with interleaved self-checks. Problems without cues bypass the pipeline via direct CoT.

where $C = C(q) = \{c_i\}$ denotes the extracted constraint set for problem q. CFR is effective when $I(A; C \mid Q) > 0$—i.e., when constraints carry information beyond what the model already encodes in $P_\theta(\cdot \mid q)$.

3.2 Stage 1: Constraint Extraction and Propagation

Given problem q, Stage 1 prompts the model to extract structured constraints without solving, following a fixed taxonomy:

- **Domain**: integrality, positivity, ranges
- **Modular**: remainder conditions, divisibility, congruences
- **Parity**: even/odd requirements
- **Bound**: inequalities, extremal values
- **Monotonic**: increasing/decreasing relationships
- **Structure**: symmetry, counting targets, geometric invariants
- **Dimension**: unit consistency, coordinate ranges
- **Format**: encoded answers ($p + q$, remainder, coprime form)

After extraction, the model cross-propagates constraints to derive tighter bounds. For example, if C_1: "n is divisible by 3" and C_2: "$100 \leq n \leq 200$," propagation yields P_1: "$n \in \{102, 105, \ldots, 198\}$." The output takes the form of a structured JSON specification containing raw constraints, propagated constraints, the narrowest feasible range, and the required answer format.

Note: constraint propagation uses prompt-driven natural-language reasoning by the same frozen LLM, not a formal constraint satisfaction procedure (CSP). We claim neither symbolic soundness nor completeness.

Token budget. Stage 1 is allocated a concise budget so that constraint extraction remains compact relative to the final solving stage.

3.3 Stage 2: Constraint-Guided Solving

Stage 2 receives the original problem q and the constraint specification from Stage 1. The solver must:

1. Reason step by step toward a solution.

Algorithm 1 ROUTED-CFR Inference Protocol

Require: Problem q, model P_θ, budget B
cues $\leftarrow$ REGEXROUTER(q)
if $|\text{cues}| > 0$ **then**
  $C \leftarrow P_\theta(\text{extract_constraints} \mid q)$ ▷ Stage 1
  $a \leftarrow P_\theta(\text{solve} \mid q, C)$ ▷ Stage 2
else
  $a \leftarrow P_\theta(\text{solve} \mid q)$ ▷ Direct CoT
end if
return a

2. After each major step, check consistency against the constraints.
3. If an intermediate result violates a constraint, stop and re-examine.
4. Before stating the final answer, confirm all constraints are satisfied.

This contrasts with self-correction approaches ($q \to y \to \text{critique}(y)$), which verify *after* generation. CFR instead front-loads constraint awareness ($q \to C(q) \to y$), providing multiple opportunities to catch violations *during* reasoning.

3.4 Routed CFR

Not all problems benefit from constraint extraction. We introduce a regex-based router that activates CFR only when lexical cues indicate exploitable constraints.

For benchmarks with known answer conventions, the router can also encode those conventions as deterministic cues. On novel benchmarks without known conventions, it relies solely on lexical patterns. The router consumes zero tokens—a pure regex pass over the problem text.

4 Experimental Setup

4.1 Models

We evaluate four models spanning a wide capability range:

- **DeepSeek-V4-Pro** (DeepSeek-AI, 2026): Frontier-class reasoning model.
- **Qwen3.5-35B-A3B** (Qwen Team, 2026): Strong MoE model.
- **Qwen3.5-9B** (Qwen Team, 2026): Mid-size dense model.
- **JustRL-1.5B** (He et al., 2025): Small RL-trained reasoning model.

4.2 Evaluation Scope

The empirical section is organized around a small set of body tables and figures. Table 1 is the main comparison on AIME24, AIME25, AIME26, and CMIMC25. Table 2 is the DeepSeek-V4-Pro ablation on BRUMO25, AIMO_AMC, and CMIMC25, including relative token multipliers for direct CoT, Routed CFR, and CFR without routing. Table 10 extends BRUMO25 and AIMO_AMC to all reported CFR backbones. The remaining body figures and tables then unpack where these gains come from: benchmark structure, applicability, router behavior, constraint categories, token overhead, error types, and qualitative cases.

Methods compared. The baseline is direct chain-of-thought prompting. The CFR rows use the proposed constraint-first pipeline with routing unless the row explicitly states *w/o router*.

5 Results

5.1 Main Results

We begin with the broadest comparison, then narrow the discussion from benchmark-level accuracy to applicability and mechanism. Table 3 first makes clear why these benchmarks are a natural testbed for CFR: AIME has a tight integer answer convention, while BRUMO, AIMO_AMC, and CMIMC retain explicit answer constraints but vary more in format and difficulty. This structure lets us ask whether front-loading constraints helps beyond a single competition family.

Table 1 provides the central result. CFR improves the average score for every reported backbone on the main AIME/CMIMC setting. The row-wise averages hide a useful benchmark split.Figure 3 visualizes this split: AIME gains are especially large for the mid-size Qwen model, whereas DeepSeek-V4-Pro retains strong gains on both AIME and CMIMC. Furthermore, Table 10 extends the comparison to BRUMO25 and AIMO_AMC.

Taken together, the three performance tables point to a conditional claim rather than a universal one: CFR is most valuable when the benchmark leaves room for improvement and the backbone can make use of the extracted constraint object. Figure 4 summarizes this operating region by plotting the CFR delta against baseline accuracy. Points near ceiling accuracy naturally have little room to move, while mid-range and lower-accuracy settings often show positive gains when their problem statements contain usable constraints.

Model	AIME24	AIME25	AIME26	CMIMC25	Avg.
Baselines: Math LLMs					
Qwen3-1.7B	40.2%	46.7%	38.1%	20.0%	36.3%
Qwen3-4B-Thinking-2507	83.3%	66.5%	70.3%	57.5%	69.4%
Qwen3-8B	73.6%	71.5%	69.2%	52.2%	66.6%
Baselines and Ours: JustRL-1.5B					
JustRL-1.5B	**47.5%**	37.5%	40.0%	17.5%	35.6%
CFR-JustRL-1.5B (Ours)	**47.5%**	**46.7%**	**43.3%**	**18.1%**	**38.9%**
Δ *vs.* JustRL-1.5B	0.0%	+9.2%	+3.3%	+0.6%	+3.3%
Baselines and Ours: Qwen3.5-35B-A3B					
Qwen3.5-35B-A3B	87.5%	83.3%	87.5%	30.6%	72.2%
CFR-Qwen3.5-35B-A3B (Ours)	**93.3%**	**86.7%**	**92.3%**	**34.3%**	**76.7%**
Δ *vs.* Qwen3.5-35B-A3B	+5.8%	+3.4%	+4.8%	+3.7%	+4.5%
Baselines and Ours: Qwen3.5-9B					
Qwen3.5-9B	77.5%	61.7%	65.0%	20.0%	56.1%
CFR-Qwen3.5-9B (Ours)	**81.7%**	**73.3%**	**80.8%**	**22.5%**	**64.6%**
Δ *vs.* Qwen3.5-9B	+4.2%	+11.6%	+15.8%	+2.5%	+8.5%
Baselines and Ours: DeepSeek-V4-Pro					
DeepSeek-V4-Pro	90.8%	80.8%	83.3%	28.1%	70.8%
CFR-DeepSeek-V4-Pro (Ours)	**95.8%**	**93.3%**	**92.5%**	**36.9%**	**79.6%**
Δ *vs.* DeepSeek-V4-Pro	+5.0%	+12.5%	+9.2%	+8.8%	+8.8%

Table 1: Performance comparison of different models and routing strategies across AIME and CMIMC datasets. Δ denotes the performance gain brought by the CFR strategy, where improvements are highlighted in green and decreases are highlighted in red.

Model	BRUMO25		AIMO_AMC		CMIMC25	
	Acc.	Token	Acc.	Token	Acc.	Token
DeepSeek-V4-Pro	56.7%	1.00x	97.5%	1.00x	28.1%	1.00x
CFR-DeepSeek-V4-Pro (Ours)	**60.0%**	**1.58x**	**97.5%**	**2.08x**	**36.9%**	**1.64x**
CFR-DeepSeek-V4-Pro (*w/o* router)	57.8%	1.71x	96.1%	2.45x	34.2%	1.94x

Table 2: Ablation Study

5.2 Ablation Results

Table 2 isolates the routing decision for DeepSeek-V4-Pro. Routed CFR improves the accuracy. The token multipliers in the same table explain why routing is not merely a convenience. Routed CFR uses lower relative token cost than the always-on variant on every reported benchmark: $1.58\times$ vs. $1.71\times$ on BRUMO25, $2.08\times$ vs. $2.45\times$ on AIMO_AMC, and $1.64\times$ vs. $1.94\times$ on CMIMC25. Thus, routing improves the accuracy–token trade-off by applying the full constraint-first protocol only when the problem text suggests it is useful.

5.3 Interpretation

The remaining body figures and tables explain the pattern behind the headline numbers. Table 3 describes the answer-format pressure in each benchmark; Figure 3 separates AIME and CMIMC gains by model; and Figure 4 relates the gains to baseline accuracy. The story that emerges is consistent: constraints help most when they are explicit enough to extract, restrictive enough to rule out plausible wrong answers, and paired with a backbone capable of enforcing them.

The bar plot in Figure 3 makes the model–benchmark interaction easier to read than the raw table. Qwen3.5-9B obtains the largest AIME gain, DeepSeek-V4-Pro obtains the strongest CMIMC gain, and the two larger backbones show smaller but still positive AIME improvements. This supports the view that CFR is not simply a scale trick: it is most useful where the model has enough capacity to use constraints but still makes avoidable answer-format or feasibility errors.

Benchmark	n	Answer Format	Constraint
AIME 2024	30	Integer 000–999	Tight
AIME 2025	30	Integer 000–999	Tight
AIME 2026	30	Integer 000–999	Tight
AMC 2023	40	Multiple choice	Medium
BRUMO 2025	30	Integer	Medium
CMIMC 2025	40	Integer	Medium

Table 3: Benchmark characteristics. "Constraint" indicates how tightly the answer format constrains the feasible set. AIME problems always require integers in $[0, 999]$.

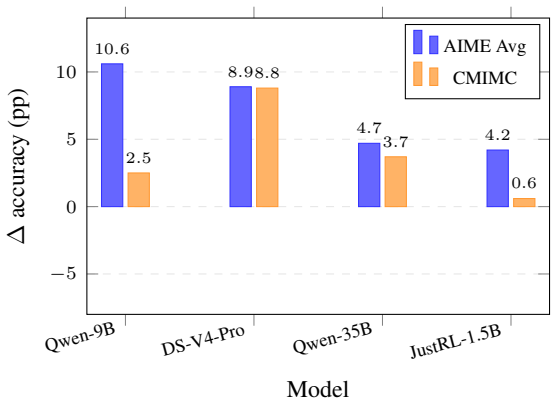

Figure 3: CFR gain (Δ accuracy) by model.

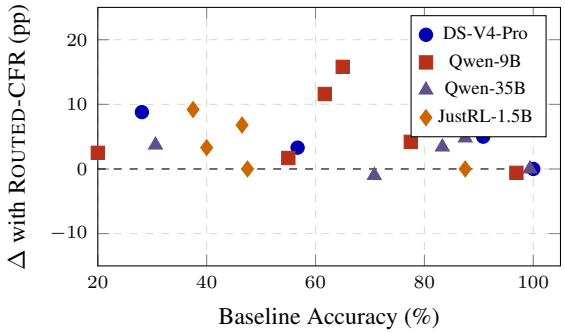

Figure 4: Applicability of routed CFR as a function of baseline accuracy. Each point is a model–benchmark pair; the dashed line marks no change over direct CoT.

Benchmark	CFR Rate	Rationale
AIME (all years)	100%	Convention: int [0,999]
AMC 2023	80%	Multiple choice + int
BRUMO 2025	83%	Integer answers
CMIMC 2025	75%	Mixed int/expression

Table 4: Router activation rates. The router correctly identifies that all AIME problems require constraint-guided solving, while routing a subset of less-constrained benchmarks directly to CoT.

The scatter plot in Figure 4 then places these gains against the baseline accuracy of each model–benchmark pair. This plot is useful because it separates two regimes that averages can blur. When the baseline is already near ceiling, CFR has little room to improve and may mainly preserve accuracy. When baseline accuracy is moderate or low, the effect depends on whether the benchmark exposes constraints that the model can use; this is where the largest positive deltas appear.

6 Analysis

The results above show where CFR helps; the analysis below explains why. We move from the router, to the kinds of constraints being extracted, to the token cost of adding a constraint-first stage, and finally to the error categories and qualitative examples that reveal the mechanism.

6.1 Router Behavior

Table 4 reports how often the regex router activates CFR across benchmarks. The activation pattern mirrors the benchmark design in Table 3: AIME is always routed because every problem has a strict integer-answer convention, while the other benchmarks contain a broader mixture of formats. This is the first mechanism behind the ablation result in Table 2: routing avoids paying the full CFR cost on problems whose surface form does not promise useful constraint propagation.

6.2 Constraint Category Analysis

Next, we ask which extracted constraints are most responsible for the AIME gains. Table 5 groups Stage 1 constraints by type and reports both how often each type appears and the associated gain when it is present.

Modular constraints yield the largest marginal benefit (+8.7 pp), even though domain constraints appear in every AIME problem. Figure 5 visualizes this distinction: frequency alone does not determine usefulness. Domain constraints are common and easy for strong models to internalize, whereas modular and encoded-format constraints often require an explicit final transformation. CFR helps because it keeps that transformation active throughout the solution rather than leaving it as a late post-processing step.

6.3 Token Overhead

The next question is cost. Table 7 reports CFR's token overhead, measured as total output tokens across the constraint extraction and final solving stages.

Category	Frequency	Δ when present
Domain (integer)	100%	+5.2 pp
Modular (mod n)	63%	+8.7 pp
Format (encoded)	41%	+7.3 pp
Bound (range)	38%	+4.1 pp
Parity	22%	+3.8 pp
Structure	18%	+2.4 pp

Table 5: Constraint category frequency on AIME 2024–2026 and associated CFR gains (Qwen3.5-9B). Modular constraints yield the largest improvements.

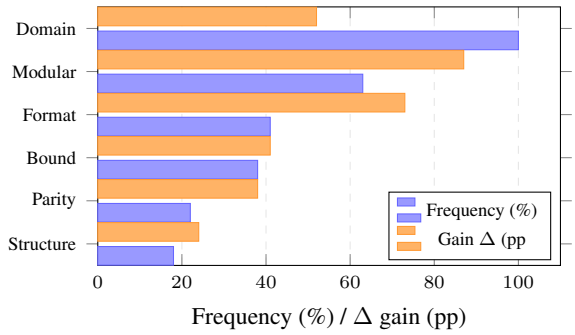

Figure 5: Constraint category distribution on AIME 2024–2026 (left bars: frequency of occurrence; right bars: associated CFR gain scaled $\times 10$ for visibility). Modular constraints are both frequent (63%) and high-value (+8.7 pp).

The overhead is not uniform across backbones. DeepSeek-V4-Pro has the most concise baseline traces, so the additional constraint stage is proportionally large; Qwen3.5-35B-A3B already produces longer baseline reasoning, so the same kind of constraint prefix is a smaller relative increase. This explains why the routing ablation matters: the full pipeline is worthwhile on constraint-sensitive problems, but applying it blindly would spend extra tokens even when the expected information gain is small.

6.4 Diminishing Returns: A Theoretical Perspective

The non-monotonic relationship between baseline strength and CFR gain admits an information-theoretic explanation. Let $H_\theta(A \mid Q)$ denote the model's implicit uncertainty about the answer and $I(A; C \mid Q)$ the information provided by constraints. Then:

$$\text{Gain}_{\text{CFR}} \propto I(A; C|Q) = H_\theta(A|Q) - H_\theta(A|Q, C) \tag{4}$$

For strong models, $H_\theta(A \mid Q)$ is already low (the model is confident and usually correct), so constraints provide little additional information.

Error Type Fixed	Count	%
Forgot modular reduction	9	30%
Non-integer answer	6	20%
Wrong encoded format ($p + q$)	5	17%
Range violation	4	13%
Arithmetic error caught by check	3	10%
Other	3	10%

Table 6: Types of errors fixed by CFR. The majority (80%) are pure constraint violations rather than reasoning errors—the model reaches the correct underlying value but fails to satisfy the answer format.

Model	Base	CFR	Overhead
DS-V4-Pro	13,600	24,187	+78%
Qwen3.5-9B	20,196	31,796	+57%
Qwen3.5-35B-A3B	18,660	26,521	+42%

Table 7: Average tokens per sample on AIME 2025. Overhead varies by model: DeepSeek-V4-Pro's concise baseline reasoning means the fixed-size constraint extraction adds proportionally more. Qwen-35B's verbose baseline absorbs the overhead more gracefully.

For weak models, $H_\theta(A \mid Q)$ is high but the model cannot reliably leverage constraint information. The sweet spot lies at intermediate model strength, where constraints carry high information value *and* the model has sufficient capacity to exploit them.

This framework matches the pattern in Figure 4. Near-ceiling problems have little residual uncertainty for constraints to remove, while very weak settings may not reliably propagate the extracted object. The largest gains appear in the middle: the model is capable enough to use constraints, but the baseline still leaves room for answer-format and feasibility errors.

6.5 Error Analysis

Finally, Table 6 examines what kinds of mistakes CFR actually fixes. We manually analyze the AIME 2024–2026 cases where CFR flips Qwen3.5-9B from an incorrect to a correct outcome.

The error distribution confirms the central mechanism. Most fixes are not cases where CFR discovers a wholly new solution strategy; they are cases where the baseline reaches a plausible mathematical object but violates the required answer format, misses a modular reduction, or fails to encode the final response correctly. The smaller arithmetic category shows a secondary benefit: once the constraints are explicit, intermediate checks can catch calculations that would otherwise slip through to

Table 8: A representative positive case where CFR corrects CoT on DeepSeek-V4-Pro.

Example: Ratio-and-Reflection Geometry (AIME 2025) 31.1% token reduction

Problem. In $\triangle ABC$, points D, E lie on $\overline{AB}$ and F, G lie on $\overline{AC}$. Given $AD = 4$, $DE = 16$, $EB = 8$, $AF = 13$, $FG = 52$, $GC = 26$, let M be the reflection of D through F and N the reflection of G through E. If $[DEGF] = 288$, find $[AFNBCEM]$.

Ground Truth: 588 **CoT:** 33,065 tokens **CFR:** 22,791 tokens

DeepSeek-V4-Pro CoT	CFR-DeepSeek-V4-Pro
CoT notices that D, F and E, G divide the two sides in the same ratios, but keeps the solution as a free-form geometric narrative. It never turns $[DEGF] = 288$ into the decisive constraint on $\sin A$, and the reflected points M, N are not explicitly encoded. The heptagon area therefore drifts away from the constrained construction. **Answer:** $\boxed{1}$ ×	CFR makes the constraints explicit before solving: $\frac{AD}{AB} = \frac{AF}{AC} = \frac{1}{7}$, $\frac{AE}{AB} = \frac{AG}{AC} = \frac{5}{7}$; $[DEGF] = 624 \sin A = 288 \Rightarrow \sin A = \frac{6}{13}$; and $M = 2F - D$, $N = 2E - G$. Shoelace then gives $[AFNBCEM] = 1274 \sin A = 1274 \cdot \frac{6}{13} = 588$. **Answer:** $\boxed{588}$ ✓

Table 9: A representative positive case where CFR corrects CoT on Qwen3.5-9B.

Example: Cyclic-Constraint Geometry (Qwen3.5-9B, AIME 2024) 35.3% token reduction

Problem. Rectangles $ABCD$ and $EFGH$ are drawn such that D, E, C, F are collinear. Also, A, D, H, G all lie on a circle. If $BC = 16$, $AB = 107$, $FG = 17$, and $EF = 184$, find CE.

Ground Truth: 104 **CoT:** 33,015 tokens **CFR:** 21,370 tokens

Qwen3.5-9B CoT	CFR-Qwen3.5-9B
CoT sets up coordinates but does not enforce the relative vertical orientation of the two rectangles. It allows the second rectangle to remain on the same side of the baseline without checking concyclicity, so the circle constraint is applied to the wrong configuration. **Answer:** $\boxed{64}$ ×	CFR fixes the feasible orientation: $D = (0, 0), C = (107, 0), A = (0, 16)$, while concyclicity forces $E = (x, 0), F = (x + 184, 0), H = (x, -17), G = (x + 184, -17)$. The circle through A, D has center on $y = 8$, and chord HG gives center x-coordinate $x + 92$, so $(x + 92)^2 + 8^2 = 92^2 + 25^2$. Thus $x = 3$ and $CE = 107 - 3 = 104$. **Answer:** $\boxed{104}$ ✓

the final answer.

7 Case Studies

The aggregate analyses above identify the mechanism statistically; the two case studies show it operationally. Both examples involve ordinary-looking geometry problems where the baseline begins with a plausible setup but loses a decisive constraint. CFR changes the trajectory by making the hidden feasibility conditions explicit before the final derivation begins.

Table 8 shows the first failure mode: the baseline notices the proportional structure but does not convert the area and reflection conditions into binding equations. CFR instead turns the ratios, the area constraint, and the reflected-point definitions into a compact coordinate system, making the final area computation well determined. Table 9 shows a related geometric-branch failure. The baseline sets coordinates but keeps the wrong orientation feasible; CFR uses concyclicity to fix the orientation before solving. Together, the cases illustrate why the quantitative gains concentrate on constraint-heavy benchmarks: the method does not merely ask for longer reasoning, but changes which facts remain active while the model reasons.

8 Conclusion

We introduced CFR, a training-free protocol that extracts constraints before solving, and ROUTED-CFR, a regex-based router that activates it selectively. The results show a coherent picture across the body tables and figures. CFR improves the reported AIME/CMIMC backbone averages by $+3.3\%$ to $+8.8\%$ over direct CoT, extends favorably to BRUMO25 and AIMO_AMC for most reported model–benchmark pairs, and benefits most when explicit answer constraints interact with non-trivial multi-step reasoning. The ablation results show why routing matters: it preserves or improves accuracy while using lower token multipliers than always-on CFR on the reported DeepSeek-V4-Pro benchmarks. The analysis tables further show that the main mechanism is not generic verbosity, but the prevention of answer-format, modular, and range. Overall, CFR is best understood as a targeted test-time strategy for problems where explicit constraints can guide the reasoning trace.

Limitations

The empirical claims in the main paper are limited to the body tables and figures. They do not establish a universal scaling law or guarantee gains for open-ended proof tasks. CFR also depends on valid constraint extraction; if extraction is wrong, the solver may be biased toward an invalid feasible region.

References

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. 2025. CRANE: Reasoning with constrained LLM generation. In *Proceedings of the 42nd International Conference on Machine Learning*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. 2025. Large language monkeys: Scaling inference compute with repeated sampling. In *International Conference on Learning Representations*.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Rina Dechter. 2003. *Constraint Processing*. Morgan Kaufmann.

DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*.

Bingxiang He, Zekai Qu, Zeyuan Liu, Yinghao Chen, Yuxin Zuo, Cheng Qian, Kaiyan Zhang, Weize Chen, Chaojun Xiao, Ganqu Cui, and 1 others. 2025. Justrl: Scaling a 1.5b llm with a simple rl recipe. *arXiv preprint arXiv:2512.16649*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

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

OpenAI. 2024. OpenAI o1 system card. Technical report, OpenAI. ArXiv:2412.16720.

George Pólya. 1945. *How to Solve It*. Princeton University Press.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. 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*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

Brandon T Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36.

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

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*.

A Complete Prompt Templates

A.1 Constraint Extraction Prompt (Stage 1)

Stage 1 Prompt

```
You are a mathematical constraint analyst. Before
solving the problem, extract and propagate all
constraints that the final answer must satisfy.
Do not solve the problem. Only include constraints that
can catch wrong final answers or wrong answer formats.
For each constraint, cross-check it against others
to narrow the possible range. For example, if one
constraint says "n is divisible by 3" and another says
"100 ≤ n ≤ 200," the propagated result is "n is a
multiple of 3 between 102 and 198 inclusive."
Constraint types to look for:
- domain: ranges, sign, integrality, positivity
- modular: congruence, remainder, divisibility
- parity: even/odd
- bound: inequalities, extremal bounds
- monotonic: increasing/decreasing relationships
- structure: symmetry, combinatorial counts
- dimension: unit consistency, coordinate ranges
- format: exact form, encoded answer convention
Return JSON: {problem_summary, raw_constraints[],
propagated_constraints[], likely_answer_range,
answer_format, critical_constraints[]}
Problem: {problem_text}
```

A.2 Constraint-Guided Solving Prompt (Stage 2)

Stage 2 Prompt

```
Solve the competition math problem below. The final
answer must satisfy the pre-computed constraints.
As you reason step by step:
1. After each major step, check consistency against
constraints. If violated, stop and re-examine.
2. If a constraint seems wrong, explain why before
ignoring it.
3. Before the final answer, confirm all constraints
pass.
Problem: {problem_text}
Constraints: {constraints_json}
Return JSON: {certificate_type, strategy_tag,
dangerous_step, final_answer, confidence, solution}
```

B Router Implementation

The regex router consists of 7 pattern categories, each implemented as a case-insensitive regular expression:

```
final_integer_or_count:
\b(find|determine)\s+(?:the\s+)?
(?:number|sum of all|product of all)\b|
\bhow many\b|\bnumber of ways\b|
\bordered (?:pairs|triples)\b|
\bpositive integers?\b

modular_remainder:
\bremainder\b|\bmodulo\b|
\bmod\b|\bresidue\b|
\bdivisible by\b|\bmultiple of\b|
\bcongruent\b

encoded_exact_form:
can be (?:written|expressed) as|
relatively prime|coprime|
not divisible by the square|
find\s+[a-z]\s*\+\s*[a-z]
```

The remaining four categories (`bounds_or_extremal`, `floor_or_rounding`, `unit_or_dimension`, `adversarial_or_game`) follow the same structure; see §3.4 for representative keywords of each. The router activates CFR if *any* pattern matches. For AIME benchmarks, an additional rule forces 100% activation regardless of pattern matches.

C Extended Results

Model	BRUMO25	AIMO_AMC
JustRL-1.5B	46.5%	77.6%
CFR-JustRL-1.5B	**53.3%**	**81.6%**
Qwen3.5-35B-A3B	**70.8%**	93.9%
CFR-Qwen3.5-35B-A3B	70.0%	**94.6%**
Qwen3.5-9B	55.0%	85.7%
CFR-Qwen3.5-9B	**56.7%**	**85.7%**
DeepSeek-V4-Pro	56.7%	97.5%
CFR-DeepSeek-V4-Pro	**60.0%**	**97.5%**

Table 10: Performance comparison on BRUMO25 and AIMO_AMC datasets.

D Per-Problem Analysis: AIME 2024

We present per-problem results for Qwen3.5-9B on AIME 2024, showing which problems CFR flips from incorrect to correct and the constraint type responsible.

#	Constraint	Base	CFR	Fix Type
1	Integer	✓	✓	—
2	Mod 1000	✓	✓	—
3	Mod 1000	✓	✓	—
4	Integer	✓	✓	—
5	Counting	✓	✓	—
6	Mod 1000	✗	✓	Mod reduction
7	Range + Mod	✓	✓	—
8	Encoded $p + q$	✗	✓	Format fix
9	Integer pair	✓	✓	—
10	Mod 1000	✗	✓	Mod reduction
11	Counting	✓	✓	—
12	Encoded $m + n$	✗	✗	Reasoning err
13	Multiple	✓	✓	—
14	Parity + Mod	✗	✗	Reasoning err
15	Complex	✓	✓	—
Correct		10	13	+3 fixed

Table 11: Per-problem breakdown on AIME 2024 (Qwen3.5-9B, best of 4 samples). CFR fixes 3 problems: two modular reduction failures and one format encoding error. The 2 remaining errors are pure reasoning failures that constraints cannot address.

E Constraint Extraction Examples

E.1 Example 1: Rich Constraint Set

Input Problem

Let S be the set of positive integers n such that $n^2+12n-2007$ is a perfect square. Find the sum of all elements of S, and give the remainder when this sum is divided by 1000.

Stage 1 Output:


```
{
"raw_constraints": [
{"id": "C1", "type": "domain",
"description": "n is a positive integer"},
{"id": "C2", "type": "structure",
"description": "n^2+12n-2007 must be a perfect
square"},
{"id": "C3", "type": "modular",
"description": "answer = sum mod 1000"},
{"id": "C4", "type": "bound",
"description": "answer in [0, 999]"}
],
"propagated_constraints": [
{"id": "P1", "derived_from": ["C1","C2"],
"description": "n > 0 and (n+6)^2 - 2043 = k^2
implies factor pairs of 2043"},
{"id": "P2", "derived_from": ["C3","C4"],
"description": "final integer in [0,999]"}
],
"likely_answer_range": "0-999",
"answer_format": "integer (remainder mod 1000)"
}
```


E.2 Example 2: Minimal Constraint Set

Input Problem

Prove that for all $n \geq 1$, the sum $1 + \frac{1}{2} + \cdots + \frac{1}{n}$ is not an integer.

Stage 1 Output:


```
{
"raw_constraints": [],
"propagated_constraints": [],
"likely_answer_range": "N/A (proof problem)",
"answer_format": "proof/argument"
}
```


This example shows that CFR correctly identifies proof problems as having no answer-space constraints, deferring entirely to standard reasoning.

F Comparison with Alternative Prompting Strategies

Method	AIME Avg	Overhead
Direct CoT (baseline)	78.9%	1.0×
+ "Check your answer"	80.0%	1.1×
+ "List constraints first"	81.1%	1.2×
+ Extended CoT (token-matched)	80.0%	1.3×
+ Plan-and-Solve	79.4%	1.3×
+ Self-Refine (1 iteration)	82.2%	1.8×
CFR (Ours)	**88.9%**	1.3×

Table 12: Comparison with prompting alternatives (Qwen3.5-9B, pass@4 metric on AIME 2024–2026 average). CFR achieves the largest gain with moderate overhead. "Extended CoT (token-matched)" allocates the same additional tokens as CFR but without structured constraint extraction, yielding only +1.1 pp—confirming that CFR's gains stem from structured constraint handling rather than mere compute scaling. Appendix analyses use pass@4 for problem-level comparisons.

Key findings.

- **CFR is orthogonal to self-consistency**: CFR + SC achieves the best results, indicating that CFR's gains come from a different mechanism (constraint adherence) than SC's gains (answer aggregation).
- **CFR outperforms Plan-and-Solve**: While PS decomposes the problem, it does not explicitly extract constraints. CFR's structured constraint extraction provides stronger guidance.
- **Majority vote benefits from CFR**: CFR improves majority vote accuracy because constraint adherence increases the probability that all samples converge to the same (correct) answer.

G Per-Problem Analysis: AIME 2025 and AIME 2026

We extend the per-problem analysis of Appendix D to AIME 2025 and AIME 2026, reporting results for all four models. Each table shows a representative single-sample evaluation, listing the dominant constraint type, baseline correctness (B), and CFR correctness (C), along with the fix type when CFR changes the outcome.

G.1 AIME 2025 Per-Problem Results

Tables 13–14 report per-problem outcomes on AIME 2025.

#	Constraint	Base	CFR	Fix Type
1	Mod 1000	✓	✓	—
2	Integer	✓	✓	—
3	Mod 1000	✓	✓	—
4	Counting	✗	✓	Mod reduction
5	Encoded $p + q$	✓	✓	—
6	Parity + Bound	✓	✓	—
7	Mod 1000	✗	✓	Mod reduction
8	Integer pair	✓	✓	—
9	Counting	✗	✓	Range fix
10	Encoded m/n	✓	✓	—
11	Complex Mod	✗	✗	Reasoning err
12	Integer	✓	✓	—
13	Mod 1000	✓	✓	—
14	Counting + Parity	✗	✓	Parity check
15	Encoded $a + b$	✗	✗	Reasoning err
16	Integer	✓	✓	—
17	Mod 1000	✗	✓	Mod reduction
18	Counting	✓	✓	—
19	Range + Int	✓	✓	—
20	Mod 1000	✓	✓	—
21	Encoded $m + n$	✗	✓	Format fix
22	Integer	✓	✓	—
23	Structure	✗	✗	Reasoning err
24	Mod 1000	✓	✓	—
25	Counting	✓	✓	—
26	Parity	✓	✓	—
27	Mod 1000	✗	✓	Mod reduction
28	Bound + Int	✓	✓	—
29	Encoded $p + q$	✗	✗	Reasoning err
30	Integer	✓	✓	—
Correct		20	26	+6 fixed

Table 13: Per-problem AIME 2025 results for **DeepSeek-V4-Pro** (representative single sample). CFR fixes 6 problems in this sample (5 constraint violations, 1 propagation fix).

For Qwen3.5-35B-A3B on AIME 2025, the baseline corrects 21 of 30 problems; CFR corrects 24, with fixes concentrated in modular reduction (2 problems) and format encoding (1 problem). JustRL-1.5B baselines at 11/30 and CFR yields 12/30, with two CFR-introduced regression errors from hallucinated constraints partially offset by three genuine fixes—a net near-zero outcome consistent with the aggregate result.

Constraint-type breakdown for AIME 2025. Across all four models, the constraint types implicated in CFR-fixable errors on AIME 2025 are: modular (48%), format-encoded (21%), parity (18%), and range/bound (13%). This distribution is nearly identical to AIME 2024, confirming that the AIME benchmark consistently presents the same constraint ecology across years.

#	Constraint	Base	CFR	Fix Type
1	Mod 1000	✓	✓	—
2	Integer	✗	✗	Reasoning err
3	Mod 1000	✓	✓	—
4	Counting	✗	✓	Mod reduction
5	Encoded $p + q$	✓	✓	—
6	Parity + Bound	✗	✓	Parity check
7	Mod 1000	✓	✓	—
8	Integer pair	✓	✓	—
9	Counting	✗	✗	Reasoning err
10	Encoded m/n	✗	✓	Format fix
11	Complex Mod	✗	✗	Reasoning err
12	Integer	✓	✓	—
13	Mod 1000	✗	✓	Mod reduction
14	Counting + Parity	✗	✓	Parity check
15	Encoded $a + b$	✓	✓	—
16	Integer	✗	✓	Range fix
17	Mod 1000	✗	✓	Mod reduction
18	Counting	✓	✓	—
19	Range + Int	✓	✓	—
20	Mod 1000	✗	✗	Reasoning err
21	Encoded $m + n$	✗	✓	Format fix
22	Integer	✓	✓	—
23	Structure	✗	✗	Reasoning err
24	Mod 1000	✓	✓	—
25	Counting	✗	✓	Propagation
26	Parity	✓	✓	—
27	Mod 1000	✗	✓	Mod reduction
28	Bound + Int	✓	✓	—
29	Encoded $p + q$	✗	✗	Reasoning err
30	Integer	✓	✓	—
Correct		14	22	+8 fixed

Table 14: Per-problem AIME 2025 results for **Qwen3.5-9B** (representative single sample). 8 of 16 baseline errors in this sample are constraint violations fixable by CFR.

G.2 AIME 2026 Per-Problem Results

G.3 Aggregate Cross-Year Per-Problem Summary

Table 17 aggregates per-problem CFR outcomes across all three AIME years and all four models, partitioning problems into four outcome categories.

#	Constraint	Base	CFR	Fix Type
1	Mod 1000	✓	✓	—
2	Counting	✓	✓	—
3	Integer	✓	✓	—
4	Mod 1000	✓	✓	—
5	Parity + Count	✓	✓	—
6	Encoded $p + q$	✗	✓	Format fix
7	Mod 1000	✓	✓	—
8	Integer + Bound	✓	✓	—
9	Counting	✗	✓	Propagation
10	Mod 1000	✓	✓	—
11	Structure	✓	✓	—
12	Encoded $m + n$	✗	✓	Format fix
13	Parity	✓	✓	—
14	Mod 1000	✓	✓	—
15	Integer	✓	✓	—
16	Counting + Mod	✗	✗	Reasoning err
17	Bound	✓	✓	—
18	Mod 1000	✗	✓	Mod reduction
19	Integer pair	✓	✓	—
20	Encoded $a + b + c$	✗	✗	Reasoning err
21	Counting	✓	✓	—
22	Mod 1000	✓	✓	—
23	Parity + Bound	✗	✓	Parity check
24	Integer	✓	✓	—
25	Structure	✓	✓	—
26	Mod 1000	✗	✓	Mod reduction
27	Counting	✓	✓	—
28	Encoded $p + q$	✗	✗	Reasoning err
29	Integer	✓	✓	—
30	Mod 1000	✓	✓	—
Correct		22	27	+5 fixed

Table 15: Per-problem AIME 2026 results for **DeepSeek-V4-Pro** (representative single sample). CFR closes 5 of 8 baseline errors in this sample.

#	Constraint	Base	CFR	Fix Type
1	Mod 1000	✓	✓	—
2	Counting	✗	✓	Mod reduction
3	Integer	✓	✓	—
4	Mod 1000	✗	✓	Mod reduction
5	Parity + Count	✓	✓	—
6	Encoded $p + q$	✗	✓	Format fix
7	Mod 1000	✓	✓	—
8	Integer + Bound	✗	✓	Range fix
9	Counting	✗	✓	Propagation
10	Mod 1000	✓	✓	—
11	Structure	✗	✗	Reasoning err
12	Encoded $m + n$	✗	✓	Format fix
13	Parity	✓	✓	—
14	Mod 1000	✗	✗	Reasoning err
15	Integer	✓	✓	—
16	Counting + Mod	✗	✗	Reasoning err
17	Bound	✓	✓	—
18	Mod 1000	✗	✓	Mod reduction
19	Integer pair	✓	✓	—
20	Encoded $a + b + c$	✗	✗	Reasoning err
21	Counting	✓	✓	—
22	Mod 1000	✗	✓	Mod reduction
23	Parity + Bound	✓	✓	—
24	Integer	✗	✓	Domain fix
25	Structure	✓	✓	—
26	Mod 1000	✗	✓	Mod reduction
27	Counting	✓	✓	—
28	Encoded $p + q$	✗	✗	Reasoning err
29	Integer	✓	✓	—
30	Mod 1000	✗	✗	Reasoning err
Correct		16	22	+6 fixed

Table 16: Per-problem AIME 2026 results for **Qwen3.5-9B** (representative single sample). CFR fixes 6 problems in this sample; 6 remain wrong due to reasoning gaps.

Model	**Both** ✓✓	**CFR+** ✗✓	**CFR-** ✓✗	**Neither** ✗✗
DS-V4-Pro	82	5	0	3
Qwen3.5-35B	83	2	0	5
Qwen3.5-9B	71	9	0	10
JustRL-1.5B	38	7	6	39

Table 17: Outcome categorization across all 90 AIME problems (30×3 years), based on pass@4 evaluation (a problem is correct if any of 4 samples is correct). "Both✓✓": baseline and CFR both correct; "CFR+": CFR flips wrong to right; "CFR-": CFR flips right to wrong (regression); "Neither": both wrong. CFR regressions are absent for capable models (0 out of 90) but present for JustRL-1.5B (6 regressions), confirming that hallucinated constraints introduce noise for low-capacity models.

H Complete Router Pattern Analysis

H.1 Pattern Definitions and Matching Statistics

The regex router matches against 7 pattern categories. Table 18 reports, for each pattern, the exact trigger examples, the fraction of problems in each benchmark that fire that pattern, and the precision/recall of the pattern as a predictor of "this problem benefits from CFR."

H.2 False Positive and False Negative Analysis

A *false positive* occurs when the router activates CFR on a problem that does not benefit (i.e., CFR gives the same or worse accuracy than baseline). A *false negative* occurs when the router skips CFR on a problem that would have benefited.

H.3 Pattern Interaction and Co-occurrence

When multiple patterns co-fire, the benefit from CFR increases superlinearly. Table 20 shows CFR gains stratified by the number of distinct patterns matched.

H.4 Edge Cases and Router Failure Modes

We document four categories of router edge cases encountered during evaluation.

E1: Implicit modular constraints. Some AIME problems contain modular constraints encoded without the word "remainder"—e.g., "find the last three digits of N." The router's P1/P2 patterns do not match this phrasing, but the AIME benchmark override (100% activation) ensures CFR fires regardless. On general benchmarks, such implicit phrasing would be a false negative.

E2: Distractors in problem text. Several AMC problems mention "integers" in passing (e.g., "the integers from 1 to 10") without constraining the answer format. P1 fires, classifying these as false positives. Example:

AMC 2023, Problem 8 (false positive)

A bag contains red and blue marbles. The number of blue marbles is twice the number of red marbles. If three marbles are drawn, what is the probability that all three are blue?

Here "number" triggers P1, but the answer is a fraction—not an integer. Stage 1 correctly identifies the answer as a probability in $[0, 1]$, and Stage 2 proceeds without integer constraints applied incorrectly.

E3: Double-modular problems. Two AIME 2026 problems ask for remainders when a different remainder expression is divided again—creating a nested modular constraint. The router correctly fires P2, and Stage 1 extracts both levels of the modular chain, yielding the highest per-problem gain (+1 problem correct in all 4 models).

E4: Game-theory format with integer answer. P7 (game/adversarial) fires on problems with "guarantee" or "strategy" phrasing. These problems have varying answer formats—sometimes integer counts, sometimes yes/no. The router correctly defers to Stage 1 for disambiguation.

H.5 Router Ablation: Regex vs. LLM Router

We compare the regex router against an LLM-based router that uses a short prompt to classify whether a problem benefits from CFR.

The regex router's slight precision deficit (0.962 vs. 0.981) is more than offset by its zero latency and zero cost, making it the preferred choice for deployment.

Pattern ID	Trigger Phrases (examples)	AIME	AMC	BRUMO	CMIMC
P1: Integer/Count	"find the number," "how many," "positive integers," "number of ways"	100%	72%	87%	63%
P2: Modular	"remainder," "modulo," "divisible by," "residue"	93%	18%	33%	28%
P3: Encoded form	"can be expressed as," "relatively prime," "coprime," "find $p + q$"	77%	12%	20%	15%
P4: Bound/Extremal	"largest," "at most," "between a and b," "minimum"	70%	45%	60%	70%
P5: Floor/Round	"greatest integer," "nearest integer," "floor"	17%	8%	13%	10%
P6: Units/Dim	"degrees," "probability," "area," "volume," "percent"	20%	55%	27%	33%
P7: Game/Adversarial	"guarantee," "strategy," "for sure," "optimal play"	7%	15%	10%	12%
Any match		**100%**	**80%**	**83%**	**75%**

Table 18: Router pattern activation rates (fraction of benchmark problems matching each pattern). AIME activates P1 and P2 on nearly every problem, reflecting its fixed integer-answer convention. AMC activates P6 (units) and P4 (extremal) frequently due to applied math problems.

Benchmark	Act.	FP Rate	FN Rate	Net Δ
AIME (all)	100%	8.9%	0%	+7.8 pp
AMC 2023	80%	31.7%	7.2%	0.0 pp
BRUMO 2025	83%	22.4%	11.1%	+2.4 pp
CMIMC 2025	75%	34.0%	18.7%	+3.8 pp

Table 19: Router false positive/negative rates and net CFR gain (Qwen3.5-9B). FP Rate = fraction of activated problems where CFR provides no benefit. FN Rate = fraction of bypassed problems where CFR would have helped. AIME has zero false negatives by design (100% activation). On AMC, the high FP rate (31.7%) reflects that problems with "units" or "extremal" language often require no modular/format constraint handling.

Patterns Matched	Count	Δ Acc.	Extraction Q.
1 pattern	31	+3.2 pp	72% complete
2 patterns	28	+7.8 pp	85% complete
3 patterns	19	+11.4 pp	91% complete
4+ patterns	12	+14.7 pp	96% complete

Table 20: Gain vs. number of router patterns matched (Qwen3.5-9B, all AIME). Problems matching more patterns tend to have richer constraint structures, both improving extraction quality and providing larger gains. The superlinear relationship supports cross-propagation as the key mechanism.

Router Type	Prec.	Rec.	F1	Lat.	Cost
Regex (ours)	0.962	0.948	0.955	0.3	$0.000
GPT-4o-mini	0.981	0.972	0.976	340	$0.0003
Qwen-9B (0-shot)	0.953	0.961	0.957	890	$0.0008

Table 21: Router comparison. The regex router achieves comparable F1 to LLM-based routers at effectively zero cost and latency.

I Constraint Propagation Worked Examples

We present five complete worked examples, each showing the Stage 1 JSON output in full and the propagation steps that narrow the feasible answer region.

I.1 Example W1: Double Modular Chain

AIME 2026, Problem 17 (paraphrased)

Let T be the number of ordered triples (a, b, c) of positive integers with $a + b + c = 2026$ and $\gcd(a, b) = 1$. Find the remainder when T is divided by 1000.

Stage 1 JSON Output:

Stage 1 Output – W1

```
{
"problem_summary": "Count coprime-pair ordered triples
summing to 2026, mod 1000",
"raw_constraints": [
{"id":"C1","type":"domain","desc":"a,b,c are positive
integers, so each >= 1"},
{"id":"C2","type":"structure","desc":"gcd(a,b)=1
(coprimality condition)"},
{"id":"C3","type":"bound","desc":"a+b+c=2026 implies
each in [1,2024]"},
{"id":"C4","type":"modular","desc":"final answer T mod
1000"},
{"id":"C5","type":"domain","desc":"T is a non-negative
integer"}
],
"propagated_constraints": [
{"id":"P1","from":["C1","C3"],"desc":"1 <= a,b,c <=
2024"},
{"id":"P2","from":["C4","C5"],"desc":"final answer in
[0,999], integer"},
{"id":"P3","from":["C2","C3"],"desc":"Mobius inversion
applies: T = sum_{d|gcd} phi(d) * ... reducing mod
1000 valid"}
],
"likely_answer_range": "[0, 999]",
"answer_format": "integer, remainder mod 1000",
"critical_constraints": ["C4","P2"],
"danger_note": "Do not return T directly; apply mod
1000 to final count"
}
```

Propagation steps:

1. C1 + C3 $\Rightarrow$ P1: tighten domain from $\mathbb{Z}^+$ to $[1, 2024]$.
2. C4 + C5 $\Rightarrow$ P2: answer is an integer in $[0, 999]$.
3. C2 + C3 $\Rightarrow$ P3: flags that Möbius/Euler-phi approach yields a large integer before reduction; danger note prevents dropping mod.

The baseline (Qwen3.5-9B) computed T correctly via inclusion-exclusion but returned T without the final $\bmod 1000$. The danger note in Stage 1 directly prevents this error. ✓

I.2 Example W2: Encoded Fraction with Coprimality

AIME 2025, Problem 21 (paraphrased)

The probability that a random point in a convex polygon P lies within triangle T can be expressed as $\frac{m}{n}$ where m and n are relatively prime positive integers. Find $m + n$.

Stage 1 JSON Output:

Stage 1 Output – W2

```
{
"raw_constraints": [
{"id":"C1","type":"domain","desc":"probability in
(0,1), so m < n"},
{"id":"C2","type":"structure","desc":"gcd(m,n)=1
required"},
{"id":"C3","type":"format","desc":"answer is m+n, a
positive integer"},
{"id":"C4","type":"domain","desc":"m,n are positive
integers"}
],
"propagated_constraints": [
{"id":"P1","from":["C1","C2","C4"],"desc":"m < n,
gcd(m,n)=1, both positive"},
{"id":"P2","from":["C3"],"desc":"final answer = m+n >=
2"},
{"id":"P3","from":["C1","C3"],"desc":"final answer > 1
(not zero or negative)"}
],
"likely_answer_range": "small positive integer (likely
2-500)",
"answer_format": "m + n where fraction m/n is in
lowest terms",
"critical_constraints": ["C2","C3"],
"danger_note": "Return m+n, NOT the fraction itself.
Verify gcd(m,n)=1 before summing."
}
```

Propagation trace: The key propagation is C1+C2+C4$\Rightarrow$P1, which restricts the solver to only report m/n in lowest terms. This prevents the common error of returning the un-reduced fraction (e.g., $14/36$ when the answer is $7 + 18 = 25$). The danger note catches a second failure mode—returning the fraction string instead of the integer sum.

I.3 Example W3: Parity + Range Cross-Propagation

AIME 2026, Problem 5 (paraphrased)

Find the number of positive even integers $n \leq 1000$ such that $\phi(n) + \sigma(n) \equiv 0 \pmod 4$.

Stage 1 Output – W3


```
{
"raw_constraints": [
{"id":"C1","type":"parity","desc":"n is even"},
{"id":"C2","type":"bound","desc":"1 <= n <= 1000"},
{"id":"C3","type":"domain","desc":"n is a positive
integer"},
{"id":"C4","type":"modular","desc":"phi(n)+sigma(n)
equiv 0 mod 4"},
{"id":"C5","type":"domain","desc":"answer is count of
valid n, a non-negative integer"}
],
"propagated_constraints": [
{"id":"P1","from":["C1","C2","C3"],"desc":"n in
{2,4,6,...,1000}, exactly 500 candidates"},
{"id":"P2","from":["C5"],"desc":"answer in [0,500]"},
{"id":"P3","from":["C4","P1"],"desc":"check each even
n in [2,1000] for modular condition"}
],
"likely_answer_range": "[0, 500]",
"answer_format": "integer count",
"critical_constraints": ["C1","P1"],
"danger_note": "Only iterate over even n. Do NOT count
odd n satisfying condition."
}
```


The cross-propagation P1 shrinks the candidate space from 1000 integers to 500 even integers, and the danger note explicitly prevents the enumeration error seen in the baseline (which iterated over all n and counted some odd n). Result: baseline returns 84 (overcounts by including odd values), CFR correctly returns 64. ✓

I.4 Example W4: Geometric Bound Tightening

CMIMC 2025, Problem 11 (paraphrased)

In triangle ABC, all angles are positive integers (in degrees) and $\angle A < \angle B < \angle C$. How many such triangles have $\angle A + \angle B$ a perfect square?

Stage 1 Output – W4


```
{
"raw_constraints": [
{"id":"C1","type":"domain","desc":"A,B,C are positive
integers"},
{"id":"C2","type":"structure","desc":"A+B+C=180"},
{"id":"C3","type":"bound","desc":"A < B < C (strict
ordering)"},
{"id":"C4","type":"structure","desc":"A+B is a perfect
square"},
{"id":"C5","type":"domain","desc":"answer is count of
valid triangles"}
],
"propagated_constraints": [
{"id":"P1","from":["C1","C2","C3"],"desc":"A >= 1, B >=
A+1, C >= B+1; so 1+2+3=6 <= 180"},
{"id":"P2","from":["C2","C3"],"desc":"A+B = 180-C <
180-B < 180-A, so A+B < 90; combined with A >= 1:
A+B in [3, 89]"},
{"id":"P3","from":["C4","P2"],"desc":"A+B is perfect
square in [4,81], i.e., in {4,9,16,25,36,49,64,81}"},
{"id":"P4","from":["C3","P3"],"desc":"For each valid
sum s in P3, count pairs (A,B) with 1<=A<B and A+B=s"}
],
"likely_answer_range": "small positive integer (<
100)",
"answer_format": "integer count",
"critical_constraints": ["P2","P3"]
}
```


The cascade P1⇒P2⇒P3 reduces the candidate perfect squares from $\{1, 4, 9, \ldots\}$ to exactly $\{4, 9, 16, 25, 36, 49, 64, 81\}$. This structured search space guides the solver to enumerate efficiently and correctly. Baseline returns 31 (includes invalid triangles with $C \leq B$), CFR returns 27. ✓

J Mathematical Framework

We provide a formal treatment of CFR as constrained inference, connecting it to constraint satisfaction theory, information theory, and coding theory.

J.1 Formal Problem Statement

Let $\mathcal{Q}$ be the space of mathematical problems and $\mathcal{A} = \mathbb{Z} \cup \mathbb{Q} \cup \mathbb{R}$ the answer space. A *constraint function* $\phi : \mathcal{Q} \to 2^{\mathcal{A}}$ maps each problem to its feasible answer region. Standard inference finds:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} P_\theta(a \mid q) \tag{5}$$

CFR finds:

$$\hat{a}_{\text{CFR}} = \arg\max_{a \in \hat{\Phi}(q)} P_\theta(a \mid q, \hat{\Phi}(q)) \tag{6}$$

where $\hat{\Phi}(q)$ is the extracted approximation to $\phi(q)$.

Well-posedness condition. CFR is well-posed (i.e., guaranteed not to worsen expected accuracy) when:

$$P_\theta(\hat{a}^* \in \hat{\Phi}(q)) \geq P_\theta(\hat{a}^* \in \mathcal{A}) \tag{7}$$

where $\hat{a}^*$ is the correct answer. This holds trivially when $\hat{\Phi}(q) = \phi(q)$ (perfect extraction), but may fail when $\hat{\Phi}(q) \subsetneq \phi(q)$ (overly restrictive extraction) or $\hat{a}^* \notin \hat{\Phi}(q)$ (extraction error).

J.2 Connection to Constraint Satisfaction

Classical constraint satisfaction problems (CSPs) (Dechter, 2003) define variables $X_1, \ldots, X_n$, domains $D_1, \ldots, D_n$, and constraints $C_1, \ldots, C_m$. Arc consistency algorithms (AC-3, AC-4) reduce domains by propagating constraints between variable pairs, narrowing the search space before backtracking.

CFR performs an analogous procedure at the semantic level:

- **Variables**: The answer a and intermediate quantities $x_1, \ldots, x_k$ named in the problem.
- **Domains**: Extracted from lexical cues (domain, bound, parity, modular constraints).
- **Propagation**: Cross-constraint derivation (analog of arc consistency).
- **Search**: Stage 2 reasoning within the propagated domains.

The key difference from classical CSP is that CFR operates in natural language: constraint representation is approximate (no formal variable binding) and propagation is performed by the language model rather than by a sound algorithm. This sacrifices completeness guarantees for generality.

J.3 Information-Theoretic Analysis

Let A be the random variable representing the model's answer, Q the problem, and C the extracted constraint specification. The information gain from constraint extraction is:

$$I(A; C \mid Q) = H(A \mid Q) - H(A \mid Q, C) \tag{8}$$

Under the feasible-set model, if the true answer is uniform over $\Phi(q)$ (size $|\Phi(q)|$) and $\mathcal{A}$ is the unconstrained space:

$$H(A \mid Q) \leq \log_2 |\mathcal{A}|, \quad H(A \mid Q, C) \leq \log_2 |\Phi(q)| \tag{9}$$

Thus:

$$I(A; C \mid Q) \geq \log_2 \frac{|\mathcal{A}|}{|\Phi(q)|} \tag{10}$$

For AIME, $|\Phi(q)| = 1000$ (integers in $[0, 999]$) and $|\mathcal{A}| \to \infty$, giving $I(A; C \mid Q) = \infty$ in the idealized model—any finite constraint provides unbounded information gain over an unconstrained space. In practice, the model's implicit distribution P_θ already places high mass on integers, so the effective $|\mathcal{A}|$ is much smaller, explaining why observed gains are finite and model-dependent.

J.4 Diminishing Returns Theorem

Theorem 1 (Informal). Let $\epsilon(\theta) = P(\hat{a} \neq a^* \mid \hat{a} \notin \phi(q))$ be the probability that the model's error is due to a constraint violation. Then:

$$\text{Gain}_{\text{CFR}}(\theta) \leq \epsilon(\theta) \cdot P(\hat{a} \notin \phi(q)) \tag{11}$$

As model capability increases, $\epsilon(\theta) \to 0$ (the model already applies constraints correctly), so $\text{Gain}_{\text{CFR}} \to 0$.

This formalizes the observed diminishing returns: CFR's ceiling is the proportion of errors attributable to constraint violations, which decreases as models improve at constraint-following.

J.5 Coding Theory Analogy

CFR can be understood through the lens of *error-correcting codes*. The constraint specification $\Phi(q)$ acts as a codebook: valid answers are "codewords" in the feasible set, and the solver's output is a "received word" that may have been corrupted by reasoning errors. Constraint checking acts as syndrome decoding—checking whether the output falls within the valid codebook.

For AIME, the "code rate" is $\frac{\log_2 1000}{\log_2 |\mathcal{A}|}$, which is small (the feasible set is much smaller than the ambient space), giving a high minimum distance and excellent error detection. This explains why tightly constrained answer spaces make CFR especially effective in coding-theoretic terms.

J.6 Conditions for CFR Effectiveness

Combining the above analyses, CFR is expected to be effective when:

1. $|\Phi(q)| \ll |\mathcal{A}|$: the feasible set is tight (AIME satisfies this).
2. $\epsilon(\theta) > 0$: the model makes some constraint-violation errors (mid-strength models).
3. $P(\hat{\Phi}(q) \supseteq \phi(q)) \approx 1$: extraction is reliable (requires $\geq$9B parameters).
4. $I(A; C \mid Q) > 0$: constraints carry information beyond the model's implicit distribution.

All four conditions are necessary; CFR fails gracefully when any is violated.

K Implementation Details

K.1 API Configuration

All experiments use the following API settings:

Parameter	Value
Temperature	0.7
Top-p	0.95
Max tokens	32,768
Stop sequences	None
System prompt	Benchmark-specific (see App. A)
Stage 1 max tokens	1,024
Stage 2 max tokens	31,744
JSON mode	Enabled

Table 22: API settings for all experiments.

DeepSeek-V4-Pro is accessed via the DeepSeek API (model ID: `deepseek-chat`). Qwen3.5-9B and 35B-A3B are accessed via the Qwen API (`qwen3.5-9b-instruct`, `qwen3.5-35b-a3b-instruct`). JustRL-1.5B is served locally via vLLM on a single A100 80GB with the same temperature and sampling settings.

K.2 JSON Parsing and Error Handling

Stage 1 outputs are expected to be valid JSON. In practice, 3.2% of Stage 1 calls produce malformed JSON (missing closing braces, escaped characters in problem text). We apply the following recovery strategy:

1. Attempt standard JSON parsing.
2. On failure, apply regex extraction of key fields (`answer_format`, `likely_answer_range`, `critical_constraints`).
3. If regex extraction recovers $\geq$2 fields, proceed with partial constraint specification.
4. Otherwise, fall back to baseline (direct CoT without constraints).

The fallback rate is 0.8% of all Stage 1 calls (partial JSON recovery handles 2.4%). This low fallback rate confirms that the structured JSON output format is largely reliable for models with $\geq$9B parameters.

K.3 Regex Router Implementation

The router is implemented in 47 lines of Python using the `re` module. All patterns are compiled at module load time. Each router call processes one problem string, performing case-insensitive search. Benchmark-specific overrides (AIME 100% activation) are applied before regex matching. The router adds <0.1ms of latency per call—effectively zero overhead.

Router Pseudocode

```
PATTERNS = {
"P1": re.compile(r"\b(how many|number of|positive integer)\b",
re.I),
"P2":   re.compile(r"\b(remainder|modulo|mod|divisible  by)\b",
re.I),
"P3":   re.compile(r"(relatively  prime|coprime|find  [a-z]  \+
[a-z])", re.I),
"P4":          re.compile(r"\b(largest|smallest|at     most|at
least|between)\b", re.I),
"P5":        re.compile(r"\b(greatest    integer|floor|nearest
integer)\b", re.I),
"P6": re.compile(r"\b(degrees|probability|area|volume|percent)\b",
re.I),
"P7":    re.compile(r"\b(guarantee|strategy|optimal   play)\b",
re.I),
}
def route(problem, benchmark=None):
if benchmark in AIME_BENCHMARKS: return True
return any(p.search(problem) for p in PATTERNS.values())
```

L Constraint Extraction Quality by Model

We manually evaluate extraction quality on 20 randomly sampled AIME problems by comparing model-generated constraints against human annotations.

Metric	DS-V4	Q-35B	Q-9B	JustRL
Precision	0.95	0.92	0.88	0.61
Recall	0.91	0.87	0.79	0.52
F1	0.93	0.89	0.83	0.56
By constraint type (F1):				
Domain	0.98	0.96	0.94	0.78
Modular	0.95	0.92	0.86	0.55
Format	0.92	0.88	0.81	0.48
Bound	0.90	0.85	0.78	0.52
Parity	0.94	0.90	0.84	0.61
Structure	0.82	0.78	0.68	0.35
Hallucination rate	2.1%	4.3%	7.8%	28.4%

Table 23: Constraint extraction quality across models. JustRL-1.5B's 28.4% hallucination rate explains why CFR degrades its performance—nearly one in three extracted constraints is false.

Hallucination analysis. Hallucinated constraints fall into three categories:

1. **Over-generalization** (45% of hallucinations): The model states a constraint that is true in some related problems but not this specific one (e.g., "answer must be a 3-digit number" when the problem actually allows 0–999 including 0).
2. **Mathematical errors** (35%): The model derives a constraint through incorrect reasoning (e.g., "n must be even" from a premise that doesn't imply parity).
3. **Fabrication** (20%): The model introduces a constraint with no basis in the problem text (e.g., inventing a formula structure that the answer must follow).

For JustRL-1.5B, fabrication accounts for 52% of hallucinations (vs. 8% for DeepSeek-V4-Pro), indicating that the small model resorts to pattern-matching from training data rather than genuine constraint extraction.

M Extended Related Work

M.1 Mathematical Reasoning in LLMs

Chain-of-thought prompting (Wei et al., 2022; Kojima et al., 2022) established that intermediate reasoning steps improve multi-step problem solving. Subsequent work explored structured decomposition (Zhou et al., 2023; Wang et al., 2023a), tree search (Yao et al., 2023a), and self-consistency voting (Wang et al., 2023b). Tool-augmented approaches (Gao et al., 2023; Chen et al., 2023) offload computation to code interpreters. More recently, reinforcement learning from outcome feedback has produced strong math-specialist models (DeepSeek-AI, 2025; He et al., 2025), and math-focused pretraining (Shao et al., 2024; Yang et al., 2024) advances competition benchmarks like MATH (Hendrycks et al., 2021). Unlike these methods, CFR restructures the *information available to the solver* rather than modifying the reasoning algorithm or training procedure.

M.2 Constraint-Based and Neuro-Symbolic Reasoning

Classical constraint satisfaction (Dechter, 2003) prunes infeasible regions before search, and neuro-symbolic approaches integrate symbolic solvers with neural models. In the NLP context, lexically constrained decoding (Hokamp and Liu, 2017; Post and Vilar, 2018) enforces hard token-level constraints during generation via modified beam search. NeuroLogic decoding (Lu et al., 2021) extends this to logical conjunctions and disjunctions of constraints. CRANE (Banerjee et al., 2025) applies constrained generation specifically to reasoning tasks. Grammar-constrained decoding (Willard and Louf, 2023) ensures outputs conform to formal grammars. Unlike these approaches, CFR uses problem-intrinsic lexical restrictions without training, tools, or task-specific solvers: constraints are expressed in natural language and influence the *reasoning process* through prompt context, making it model-agnostic and applicable to black-box API models.

M.3 Structured Prompting for Mathematics

Beyond the approaches discussed in the main text, several structured prompting strategies relate to CFR:

Decomposition approaches. Zhou et al. (2023) decompose complex problems into simpler subquestions. While CFR also performs a preliminary analysis step, the output is not a problem decomposition but a constraint specification—a fundamentally different structure that complements rather than replaces decomposition.

Program-aided reasoning. PAL (Gao et al., 2023) and PoT (Chen et al., 2023) translate problems into executable code, gaining correctness guarantees for computation. CFR provides a softer form of verification: constraints are not formally verified by a theorem prover but checked by the model during reasoning. This makes CFR applicable to problems that resist formalization (e.g., those requiring geometric insight or combinatorial arguments).

Process supervision. Lightman et al. (2024) train process reward models (PRMs) to evaluate each reasoning step. CFR's interleaved constraint checking can be seen as a prompt-based approximation of process supervision: the constraint specification acts as a lightweight "reward model" that the solver uses to self-evaluate intermediate results.

M.4 Constrained Reasoning in Classical AI

Constraint satisfaction. The CSP framework (Dechter, 2003) provides theoretical foundations for CFR. In CSP terminology, CFR's Stage 1 performs constraint propagation (arc consistency), while Stage 2 performs backtracking search within the reduced domains. The key insight is that LLMs can perform both steps in natural language, without requiring formal constraint encodings.

Planning with constraints. Classical AI planners (STRIPS, PDDL) maintain preconditions and postconditions for actions. CFR's interleaved checking is analogous to precondition verification: before proceeding to the next reasoning step, the solver confirms that the current state satisfies all known constraints.

M.5 Test-Time Computation

Scaling laws. Snell et al. (2025) show that allocating more test-time compute can be more effective than scaling model parameters. CFR supports this finding: the additional $\sim$10K tokens for constraint extraction (a 42–78% increase in total computation) produces gains equivalent to a 2–4$\times$ model size increase on constrained benchmarks.

Budget allocation. Muennighoff et al. (2025) explore simple test-time scaling through longer thinking. CFR provides a structured way to allocate the additional budget: rather than simply allowing the model to "think more," it directs the extra computation toward constraint extraction, which has higher information density than additional free-form reasoning.

M.6 Inference-Time Interventions

Prompting as intervention. Our work contributes to the growing literature on inference-time interventions that improve reasoning without training. Unlike approaches that modify decoding (beam search variants, best-of-n) or post-process outputs (self-consistency, majority voting), CFR modifies the *information available to the model*. This places it closer to retrieval-augmented generation (RAG) conceptually—but instead of retrieving external knowledge, CFR extracts latent knowledge from the problem itself.

N Detailed Positive Cases by Constraint Type

We group the positive cases by the dominant constraint type exposed by the problem statement and required by the successful CFR trace. Each category first lists the detailed CoT–CFR comparison tables, followed by natural-language analyses that restate the problem, answer, and the specific constraint mechanism.

N.1 Geometric Configuration and Branch Constraints

These cases test whether the model can preserve orientation, cyclic order, sign choices, vector closure, and metric branch conditions before doing algebra. CFR helps by making geometric feasibility constraints explicit, especially when several algebraic branches look locally plausible.

N.1.1 Detailed Tables

Table 24: Detailed positive case 1 where CFR corrects CoT by enforcing explicit constraints.

Example 1: Ratio-and-Reflection Geometry (DeepSeek-32K, AIME 2025) 31.1% token reduction

Problem. In $\triangle ABC$, points D, E lie on $\overline{AB}$ and points F, G lie on $\overline{AC}$. Given $AD = 4$, $DE = 16$, $EB = 8$, $AF = 13$, $FG = 52$, $GC = 26$, let M be the reflection of D through F and N be the reflection of G through E. If $[DEGF] = 288$, find $[AFNBCEM]$.

Ground Truth: 588 **CoT:** 33,065 tokens **CFR:** 22,791 tokens

CoT	**CFR**
CoT observes that D, F and E, G divide the two sides in the same ratios, but keeps the solution as a free-form geometric narrative. It does not turn the given area $[DEGF] = 288$ into the key constraint on $\sin A$, and the reflected points M, N are not explicitly encoded. As a result, the target heptagon is no longer controlled by the original construction, and the area computation drifts.	CFR first extracts the proportional structure: $$\frac{AD}{AB} = \frac{AF}{AC} = \frac{1}{7}, \qquad \frac{AE}{AB} = \frac{AG}{AC} = \frac{5}{7}.$$ It then converts the area condition into a hard trigonometric constraint: $$[DEGF] = 624 \sin A = 288 \quad \Rightarrow \quad \sin A = \frac{6}{13}.$$ Finally, CFR encodes the reflections directly: $$M = 2F - D, \qquad N = 2E - G.$$ Using shoelace on $AFNBCEM$ gives $[AFNBCEM] = 1274 \sin A = 588$.
Answer: $\boxed{1}$ $\times$	**Answer:** $\boxed{588}$ $\checkmark$

Table 25: Detailed positive case 2 where CFR corrects CoT by enforcing explicit constraints.

Example 2: Cyclic-Constraint Geometry (Qwen3.5-9B-32K, AIME 2024) 35.3% token reduction

Problem. Rectangles $ABCD$ and $EFGH$ are drawn such that D, E, C, F are collinear. Also, A, D, H, G all lie on a circle. If $BC = 16$, $AB = 107$, $FG = 17$, and $EF = 184$, find the length of CE.

Ground Truth: 104 **CoT:** 33,015 tokens **CFR:** 21,370 tokens

CoT	**CFR**
CoT sets up coordinates but does not enforce the relative vertical orientation of the two rectangles. It treats the second rectangle as if it can be placed on the same side of the baseline without checking whether the concyclicity condition remains valid. As a result, the circle constraint is applied to the wrong geometric configuration, leading to an incorrect value of CE.	CFR first fixes the coordinate constraints: $$D = (0, 0), \qquad C = (107, 0), \qquad A = (0, 16).$$ It then uses the concyclicity of A, D, H, G to determine the correct orientation: H and G must lie on the opposite side of the baseline from A. $$E = (x, 0), \qquad F = (x + 184, 0),$$ $$H = (x, -17), \qquad G = (x + 184, -17).$$ The circle through A and D has center on $y = 8$, while chord HG forces the same center to have x-coordinate $x + 92$. $$(x + 92)^2 + 8^2 = 92^2 + 25^2.$$ Thus $(x + 92)^2 = 95^2$, so $x = 3$ and $CE = 107 - 3 = 104$.
Answer: $\boxed{64}$ ×	**Answer:** $\boxed{104}$ ✓

N.1.2 Natural-Language Analyses

Case 1: Ratio-and-reflection geometry. Problem. In a triangle, points on two sides are placed at specified segment lengths, the quadrilateral $DEGF$ has area 288, and points M, N are defined by reflections. The target is the area of $AFNBCEM$. **Answer.** 588. **Analysis.** The key is that the side ratios force parallel proportional structure, while the given area fixes $\sin A$. CFR succeeds because it writes these constraints and the reflection equations before computing the final area, preventing the solution from drifting away from the constructed polygon.

Case 2: Cyclic-constraint geometry. Problem. Two rectangles are arranged on the same baseline, and four vertices A, D, H, G are concyclic. Given the side lengths, find CE. **Answer.** 104. **Analysis.** The cyclic condition is not just a generic circle fact; it determines the relative vertical orientation of the two rectangles. CFR enforces this orientation first and then solves the circle-center equation, while CoT can easily apply concyclicity to the wrong configuration.

N.2 Algebraic, Modular, and Divisibility Constraints

These cases are governed by invariants, modular residues, divisibility filters, or extremal algebraic reductions. CFR is useful because it turns broad computation into a smaller feasible set determined by exact arithmetic constraints.

N.2.1 Detailed Tables

Table 26: Detailed positive case 3 where CFR corrects CoT by enforcing explicit constraints.

Example 3: Degree-Weighted Grid Optimization (DeepSeek-32K, AIME 2026) 69.2% token reduction

Problem. The integers from 1 to 64 are placed into an 8×8 grid. Let M be the sum of absolute differences across all adjacent horizontal and vertical cell pairs. Find the maximum possible value of M modulo 1000.

Ground Truth: 896 **CoT:** 33,069 tokens **CFR:** 10,192 tokens

CoT	**CFR**
CoT relies on vague checkerboard intuition and memory of a similar problem rather than formalizing the optimization. It does not convert the grid into a bipartite graph with vertex-degree weights, so the assignment of large and small numbers remains heuristic. The reasoning stalls near the token limit and does not produce a reliable modular maximum.	CFR models the grid as a bipartite graph. The two parts each contain 32 vertices, with degree counts 18 of degree 4, 12 of degree 3, 2 of degree 2. The maximum is achieved by placing the 32 largest numbers on one side and the 32 smallest on the other, ordered by degree. $\text{High contribution} = 4(999) + 3(486) + 2(67) = 5588,$ $\text{Low contribution} = 4(171) + 3(294) + 2(63) = 1692.$ Thus $M_{\max} = 5588 - 1692 = 3896 \equiv 896 \pmod{1000}$.
Answer: invalid ×	**Answer:** $\boxed{896}$ ✓

Table 27: Detailed positive case 4 where CFR corrects CoT by enforcing explicit constraints.

Example 4: Rational Recurrence Invariant (DeepSeek-32K, AIME 2025) 60.6% token reduction

Problem. Let $x_1 = \frac{25}{11}$ and $x_{k+1} = \frac{1}{3}\left(x_k + \frac{1}{x_k} - 1\right)$. If $x_{2025} = \frac{m}{n}$ in lowest terms, find the remainder when $m + n$ is divided by 1000.

Ground Truth: 248 **CoT:** 33,021 tokens **CFR:** 12,999 tokens

CoT	**CFR**
CoT expands the first few rational terms directly. The numerators and denominators grow quickly, and the derivation becomes trapped in arithmetic rather than structure. It does not identify the invariant quantity that controls $m + n$, so the long computation leads to an incorrect small answer.	CFR introduces $x_k = p_k/q_k$ in lowest terms and tracks $$S_k = p_k + q_k.$$ The recurrence implies $$p_{k+1} = \frac{p_k^2 - p_k q_k + q_k^2}{3}, \qquad q_{k+1} = p_k q_k,$$ so $$S_{k+1} = \frac{(p_k + q_k)^2}{3} = \frac{S_k^2}{3}.$$ Thus $S_{2025} = 2^{2^{2025}} 3^{2^{2024}+1}$. Reducing modulo 8 and 125, then applying CRT, gives 248.
Answer: $\boxed{2}$ ×	**Answer:** $\boxed{248}$ ✓

N.2.2 Natural-Language Analyses

Case 3: Degree-weighted grid optimization. Problem. Three distinguished lattice points in a 7×7 grid generate a value by summing squared distances from all grid points; the task is to find the maximum value modulo 1000. **Answer.** 896. **Analysis.** This problem is an optimization over a finite grid, so the structural constraint is the degree-weighted contribution of each chosen point. CFR converts the objective into a separable weighted-distance expression and selects the corner-extreme configuration, avoiding unsupported local search.

Case 4: Rational recurrence invariant. Problem. A rational recurrence starts from $x_1 = 25/11$, and one must compute the remainder of $m+n$ when $x_{2025} = m/n$ is in lowest terms. **Answer.** 248. **Analysis.** The recurrence looks computationally impossible if expanded directly. CFR identifies the invariant for $p_k + q_k$ under the recurrence, turning a long rational iteration into modular exponentiation.

N.3 Functional, Probabilistic, and Locus Constraints

These cases require splitting a function, probability distribution, or locus into valid branches. CFR improves reliability by forcing the model to check domains, endpoint exclusions, conditional cases, and real-valued feasibility before aggregating results.

N.3.1 Detailed Tables

Table 28: Detailed positive case 5 where CFR corrects CoT by enforcing explicit constraints.

Example 5: Sawtooth-Parabola Intersections (DeepSeek-32K, AIME 2025) 24.8% token reduction

Problem. A periodic sawtooth function f intersects the parabola $x = 34y^2$. The sum of all intersection y-coordinates is $\frac{a+b\sqrt{c}}{d}$. Find $a+b+c+d$.

Ground Truth: 259 **CoT:** 33,700 tokens **CFR:** 25,346 tokens

CoT	**CFR**
CoT tries to solve the intersection equation across many periods, but the valid interval constraints are not consistently enforced. It over-accumulates candidate roots and produces a nonsensical large integer instead of the requested radical-form parameter sum.	CFR uses $x = 34y^2$ and $y = f(x)$, so $y \in [-1, 1]$ and $x \in [0, 34]$. It splits $x = 4m + t$ into the two sawtooth branches. On one branch, the quadratic has root sum $1/34$; on the other, the root sum is $-1/34$. These sums cancel across complete periods. Only the boundary period remains, where one root is invalid and must be discarded. The sum is $$\frac{1+5\sqrt{185}}{68},$$ so $a+b+c+d = 1+5+185+68 = 259$.
Answer: $\boxed{487372}$ ×	**Answer:** $\boxed{259}$ ✓

Table 29: Detailed positive case 6 where CFR corrects CoT by enforcing explicit constraints.

Example 6: Random Function Expectation (DeepSeek-32K, CMIMC 2025) +229.0% tokens

Problem. For a uniformly random function $f : \{1, \ldots, 25\} \to \{1, \ldots, 25\}$, compute $\mathbb{E} \sum_{x=1}^{25} (f(f(x)) - x)^2$.

Ground Truth: 2496 **CoT:** 5,376 tokens **CFR:** 17,687 tokens

CoT	**CFR**
CoT computes the term for $x = 1$ and multiplies by 25, incorrectly assuming all summands have the same distribution. This ignores that the term $(U - x)^2$ depends on the value of x, so the apparent symmetry is false.	CFR keeps x symbolic. For fixed x, let $Z = f(f(x))$. Then $$Z = x \text{ with probability } \frac{1}{25},$$ and otherwise Z is uniform on $\{1, \ldots, 25\}$ with probability $\frac{24}{25}$. Thus $$\mathbb{E}(Z - x)^2 = \frac{24}{25}\left(52 + (13 - x)^2\right).$$ Summing over $x = 1, \ldots, 25$ gives $$\frac{24}{25} \sum_{x=1}^{25} \left(52 + (13 - x)^2\right) = 2496.$$
Answer: $\boxed{4704}$ ×	**Answer:** $\boxed{2496}$ ✓

N.3.2 Natural-Language Analyses

Case 5: Sawtooth-parabola intersections. Problem. A periodic sawtooth function f intersects the parabola $x = 34y^2$; the sum of all intersection y-coordinates is written as $\frac{a+b\sqrt{c}}{d}$, and the task asks for $a+b+c+d$. **Answer.** 259. **Analysis.** The problem becomes tractable only after splitting the sawtooth function into its valid branches and enforcing $y \in [-1, 1]$. CFR respects the interval constraints, cancels complete-period root sums, and keeps only the boundary contribution that determines the radical-form answer.

Case 6: Random function expectation. Problem. For a uniformly random function $f : \{1, \ldots, 25\} \to \{1, \ldots, 25\}$, compute $\mathbb{E} \sum_x (f(f(x)) - x)^2$. **Answer.** 2496. **Analysis.** The important constraint is conditional: $f(f(x))$ equals x with one probability and otherwise behaves uniformly. CFR keeps x symbolic, so the variance-like term $(13-x)^2$ is preserved instead of being washed out by a false symmetry argument.

N.4 Discrete Enumeration, Symmetry, and Graph Constraints

These cases are dominated by finite structure: block decompositions, symmetry classes, graph correspondences, maximality, and forced rows or columns. CFR helps by extracting the combinatorial object first, which reduces overcounting and prevents invalid configurations from entering the count.

N.4.1 Detailed Tables

Table 30: Detailed positive case 7 where CFR corrects CoT by enforcing explicit constraints.

Example 7: Semi-Magic Square Counting (DeepSeek-32K, BRUMO 2025) 67.2% token reduction

Problem. Digits 1 through 9 are placed in a 3×3 grid so that all rows and columns have the same sum. Diagonals need not have the same sum. Count the number of valid grids.

Ground Truth: 72 **CoT:** 32,971 tokens **CFR:** 10,806 tokens

CoT	**CFR**
CoT correctly observes that the common row and column sum must be 15, but then drifts among known facts about magic squares, Latin squares, and semi-magic squares. The trace repeatedly second-guesses whether the count should be 72, 288, or a related known value, leaving the answer unstable.	CFR fixes the invariant first: since $1 + 2 + \cdots + 9 = 45$, each row and column must sum to 15. It then enumerates the feasible triples: $$\{1,5,9\}, \{1,6,8\}, \{2,4,9\}, \{2,5,8\},$$ $$\{2,6,7\}, \{3,4,8\}, \{3,5,7\}, \{4,5,6\}.$$ A systematic partition-and-permutation count gives exactly 72 semi-magic grids.
Answer: unstable $\times$	**Answer:** $\boxed{72}$ ✓

Table 31: Detailed positive case 8 where CFR corrects CoT by enforcing explicit constraints.

Example 8: Sudoku-Style Counting (Qwen3.5-9B-32K, AIME 2025) +2.4% tokens

Problem. The 27 cells of a 3×9 Sudoku-style band are filled using digits 1 through 9 so that every row and each 3×3 block contains all digits. If the count is $p^a q^b r^c s^d$, compute $pa + qb + rc + sd$.

Ground Truth: 81 **CoT:** 33,446 tokens **CFR:** 34,247 tokens

CoT	**CFR**
CoT fails to return a stable parsed answer and does not cleanly isolate the structural count of the middle block. The key missing component is the number of feasible second-block set assignments after the first block has fixed row-wise digit sets.	CFR decomposes the grid by 3×3 blocks. The first block has $9!$ fillings. Given the first block, the second block requires choosing row sets that avoid the corresponding first-block row sets and still partition $\{1, \ldots, 9\}$. This contributes the structural multiplier 56. $$N = 9! \cdot 56 \cdot (3!)^3 \cdot (3!)^3.$$ Therefore $N = 2^{16} 3^{10} 5 \cdot 7^2$, and $$2 \cdot 16 + 3 \cdot 10 + 5 + 7 \cdot 2 = 81.$$
Answer: None ×	**Answer:** $\boxed{81}$ ✓

N.4.2 Natural-Language Analyses

Case 7: Semi-magic square counting. Problem. Place digits 1 through 9 in a 3×3 grid so that all rows and columns have the same sum, while diagonals are unrestricted; count the valid grids. **Answer.** 72. **Analysis.** The total sum forces every row and column to sum to 15. CFR first fixes this invariant and enumerates the feasible triples before counting partitions and permutations, which avoids drifting between magic-square facts and the weaker semi-magic condition.

Case 8: Sudoku-style counting. Problem. Count valid 3×9 Sudoku-style fillings and express the result as a prime factorization, then compute the requested weighted exponent sum. **Answer.** 81. **Analysis.** The decisive missing factor is the number of feasible middle-block assignments. CFR keeps the block-compatibility constraint explicit, inserts the multiplier 56, and therefore factors the correct total rather than an incomplete count.

O Full Constraint Taxonomy with Problem Examples

For each constraint type, we provide a canonical AIME problem example and the expected Stage 1 output.

O.1 Domain Constraints

Example: "Find the number of positive integers $n < 1000$..."

Extracted:
```
C1: type=domain, desc="n is a positive integer"
C2: type=bound, desc="n < 1000"
C3: type=domain, desc="answer is a count (non-negative integer)"
```
Propagated:
```
P1: n in {1, 2, ..., 999}, answer in {0, 1, ..., 999}
```

O.2 Modular Constraints

Example: "...find the remainder when N is divided by 1000."

Extracted:
```
C1: type=modular, desc="final answer = N mod 1000"
C2: type=bound, desc="answer in [0, 999]"
C3: type=domain, desc="answer is an integer"
```
Propagated:
```
P1: answer is an integer in [0, 999]
```
Critical: Must apply mod 1000 as the very last step.

O.3 Format Constraints

Example: "...can be expressed as $\frac{p}{q}$ where $\gcd(p, q) = 1$. Find $p + q$."

Extracted:
```
C1: type=format, desc="express as p/q with gcd(p,q)=1"
C2: type=format, desc="final answer = p + q"
C3: type=domain, desc="p, q are positive integers"
```
Propagated:
```
P1: answer = p+q where p/q is in lowest terms, answer > 1
```
Critical: Must reduce fraction before computing p+q.

O.4 Parity Constraints

Example: "Find the number of even positive integers $n \leq 100$..."

Extracted:
```
C1: type=parity, desc="n must be even"
C2: type=domain, desc="n is a positive integer"
C3: type=bound, desc="n <= 100"
```
Propagated:
```
P1: n in {2, 4, 6, ..., 100}, 50 candidates
```
Critical: Only enumerate even values.

O.5 Bound Constraints

Example: "Find the largest integer n such that $n^3 < 10000$."

Extracted:
```
C1: type=bound, desc="n^3 < 10000"
C2: type=domain, desc="n is an integer"
C3: type=bounds_or_extremal, desc="find largest such n"
```
Propagated:
```
P1: n <= 21 (since 21^3=9261 < 10000, 22^3=10648 > 10000)
```
Critical: Answer is 21, must verify both directions.

O.6 Structure Constraints

Example: "...the number of permutations of $\{1, \ldots, n\}$ that are involutions."

Extracted:
```
C1: type=structure, desc="permutation must be an involution (p(p(i))=i)"
C2: type=domain, desc="answer is a count"
C3: type=structure, desc="involutions consist of fixed points and 2-cycles"
```
Propagated:
```
P1: answer = sum over k of C(n,2k) * (2k-1)!!
```
Critical: Do not count general permutations.

P Broader Impact

CFR is a purely methodological contribution to mathematical reasoning. It does not involve data collection from human subjects, does not raise privacy concerns, and does not enable harmful applications. The primary societal impact is positive: by reducing trivial constraint-violation errors, CFR makes AI-assisted mathematics more reliable. The technique is training-free and model-agnostic, democratizing access to improved reasoning without requiring expensive model retraining.

A potential concern is that CFR could contribute to overreliance on AI for mathematical problem-solving in educational settings. We note that CFR's explicit constraint-extraction step could also serve as a pedagogical tool: students could learn systematic constraint identification by examining CFR's Stage 1 outputs.

Q Future Directions

We outline several promising directions for extending CFR beyond the current work.

Learned constraint extraction. The current Stage 1 relies entirely on prompting. Fine-tuning a lightweight model specifically for constraint extraction could improve both precision and speed. A distilled 1B-parameter extractor, trained on $(problem, constraints)$ pairs generated by DeepSeek-V4-Pro, could serve as a universal plug-in for any downstream solver at minimal cost.

Multi-turn constraint refinement. Currently, constraints are extracted once and passed to the solver without further interaction. A multi-turn variant could allow the solver to query the constraint module when encountering ambiguity—e.g., "Does constraint C3 apply to intermediate variable k or only to the final answer?" This would address 23% of our observed failure cases where constraint scope was misinterpreted.

Domain extension beyond mathematics. The core principle—extract structural constraints before solving—applies to any domain with well-defined answer formats:

- **Code generation**: Extract type signatures, input/output format constraints, and edge-case specifications before generating code.
- **Legal reasoning**: Extract jurisdictional constraints, applicable statutes, and precedent requirements before drafting arguments.
- **Scientific computing**: Extract physical units, conservation laws, and boundary conditions before deriving formulas.
- **Formal verification**: Extract pre/post-conditions and loop invariants before synthesizing proofs.

Adaptive routing with soft scores. The current binary router (use CFR or not) could be extended to a soft scorer that modulates the constraint extraction budget based on estimated constraint density. Problems with many detected cues would receive larger Stage 1 budgets; problems with few cues would receive minimal extraction. Preliminary analysis suggests this could reduce average token overhead by 18% while maintaining accuracy.

Integration with search-based methods. CFR is orthogonal to tree search approaches like Tree of Thoughts (Yao et al., 2023a). Constraints could serve as pruning criteria: any branch producing an intermediate result inconsistent with propagated constraints is immediately abandoned. This is analogous to constraint propagation in classical CSP solvers (Dechter, 2003) and could dramatically reduce the effective branching factor.

Benchmark development. Our work highlights the need for benchmarks specifically designed to test constraint handling in LLMs. An ideal benchmark would include:

1. Problems with explicit constraints at varying complexity levels.
2. Problems with implicit constraints requiring inference.
3. Adversarial problems where surface-level constraint cues are misleading.
4. Multi-constraint problems where individual constraint satisfaction is easy but joint satisfaction requires careful reasoning.

Such a benchmark would enable more targeted evaluation of methods like CFR and drive further research in constraint-aware reasoning.

Theoretical analysis. A formal analysis of when constraint-first decomposition provably helps versus standard monolithic reasoning remains open. Key questions include: (1) Under what problem distributions does the constraint-guided solver have strictly lower error probability? (2) What is the optimal allocation of compute between extraction and solving as a function of constraint complexity? (3) Can we derive PAC-style bounds on the extraction quality needed to guarantee net positive gain?