WHEN 105 BECOMES 10^5: PRIOR-INDUCED LITERAL OVERRIDE IN COMPETITIVE-PROGRAMMING CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Competitive-programming constraints determine which algorithms are computationally feasible. We study a specification-faithfulness failure in which an authoritative decimal literal also resembles a fused power-of-ten expression: `105` is the integer 105 but also the surface left by deleting the exponent marker from 10^5. We call this *prior-induced literal override*. The corruption itself already appears at scale in CodeContests-O: among 11,682 problem descriptions, 3,008 (25.75%) contain at least one studied fused surface, and the matched occurrences were verified as exponent-loss cases rather than intended decimals. Holding the critical digits fixed, we vary semantic context and pair every collision with a same-length non-collision control. Across four open-weight models, attractor share is effectively zero in neutral copying but rises to 74.5% in full problem statements; unprompted greedy override reaches 64.3%. Direct literal disambiguation reduces attraction by about 50–52 percentage points, whereas generic self-verification has little effect. Across ten competitive-programming problems and six frontier code models, the collision surface `105` lowers executable performance relative to literal-scale controls and often shifts generated programs toward the regime induced by an explicit 10^5 constraint. Supporting probability, tokenization, and residual-stream analyses connect the shift to the literal–attractor competition. The failure is not repair itself, but failure to condition repair on source authority.

1 INTRODUCTION

In competitive programming, numerical constraints are part of the algorithmic specification: they determine which time and space complexities are feasible. Consider a problem whose input satisfies $n \leq 105$. A cubic dynamic program can be plausible when the bound is the decimal integer one hundred five, yet impossible when the same visible characters are reconstructed as 10^5. A model that silently inserts an exponent therefore does more than make a numerical mistake—it changes the inferred computational regime before choosing an algorithm.

Competitive programming is a particularly sharp setting because powers of ten are a strong domain convention, and the exact corruption pattern we study already occurs at scale in CodeContests-O. In an audit of all 11,682 problem descriptions (Cai et al., 2026), 1,513 (12.95%) contain the fused surface `105`, and 3,008 (25.75%) contain at least one of our seven studied surfaces. We verified that the matched occurrences are exponent-loss cases rather than intended decimal literals. Thus $10^5 \mapsto$ `105` is not a hypothetical corruption channel but an existing data-quality problem; full counts and matching rules are in Appendix A.

The central ambiguity is epistemic rather than syntactic. The decimal `105` is well formed and unambiguous under ordinary programming syntax, but it is also the residue of 10^5 after loss of a caret, superscript tag, or layout boundary. Repair is appropriate on genuinely lossy inputs; our benchmark instead studies *literal-authoritative* specifications, where the displayed characters are exact. We ask whether models condition repair on that source authority, or whether competitive-programming context still drives them toward the familiar power-of-ten reconstruction.

We study this behavior with a controlled within-string design. The critical digit sequence is held fixed while its semantic environment changes across neutral copying, mathematical questions, constraint

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

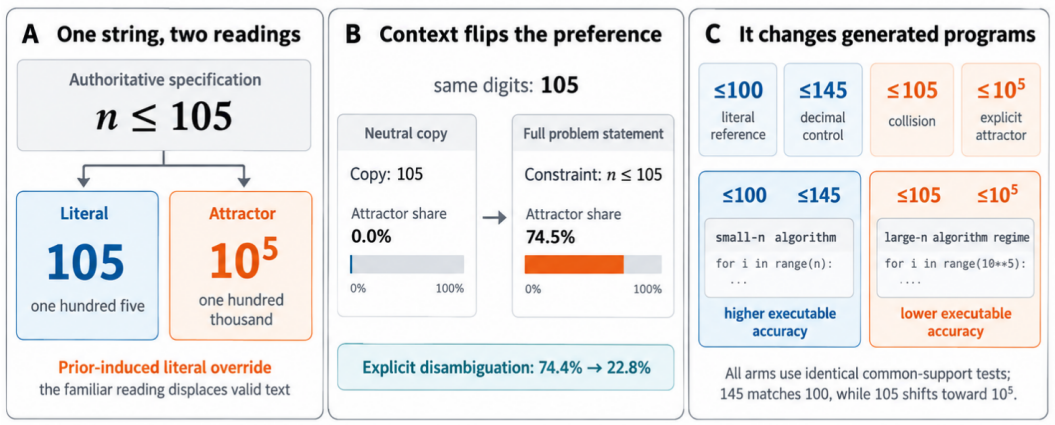

Figure 1: Overview of prior-induced literal override in competitive-programming specifications. The same authoritative surface `105` supports a literal reading (105) and a high-prior reconstruction (10^5) (A). Holding the digits fixed, competitive-programming context can flip the preference, while direct disambiguation restores the literal interpretation (B). The consequential prediction is algorithmic: if `105` is reconstructed as 10^5, generated programs should move from the small-n computational regime toward the explicit-10^5 regime rather than merely respond to a generic change in a three-digit bound (C).

blocks, short specifications, full competitive-programming statements, code/API schemas, and non-CS specifications. Every fused power-of-ten collision is paired with a same-length decimal control, and we score complete literal and attractor answers before validating the same preference under free generation. This design separates context-induced reinterpretation from copying failure, tokenization difficulty, and ordinary coding mistakes: the observed numeral does not change, only the evidence surrounding it does.

We test the vulnerability at two levels. Controlled probability and generation experiments isolate when competitive-programming context shifts a copyable literal toward the power-of-ten attractor and when direct literal evidence suppresses that shift. We then ask a separate frontier-model question: whether the same collision surface remains behaviorally consequential in current large code-generation systems. On ten matched problems, `105` moves generated programs away from the small-n reference regime and often toward the explicit-10^5 regime. Probability, tokenization, and residual-stream analyses provide supporting evidence about the literal–attractor competition (Figure 1).

Our contributions are:

- We formulate *prior-induced literal override* as a specification-faithfulness failure in competitive programming: a model applies a familiar power-of-ten repair even when an authoritative decimal should be preserved. A corpus audit finds verified exponent-loss cases matching at least one studied fused surface in 25.75% of 11,682 CodeContests-O descriptions, showing that the corruption channel itself already occurs at scale.

- We establish the effect with a controlled fixed-string evaluation over seven collision–control pairs and seven contexts. Matched controls, difference-in-differences, free generation, and interventions show that the shift is context selective and is suppressed most reliably by task-local literal evidence.

- We connect interpretation to algorithmic consequence. Across ten matched problems and six frontier code models, `105` can change the inferred feasibility regime, shift solution families, and reduce executable correctness; behavior often moves specifically toward the explicit-10^5 control. Tokenization, probability decomposition, and activation patching provide supporting evidence.

2 Related Work

Specification faithfulness and robustness in code generation. Code-generation benchmarks evaluate whether models translate natural-language specifications into executable programs (Hendrycks et al., 2021; Li et al., 2022; Jain et al., 2025). Robustness studies show that generated code can be unstable under natural-language, syntax, and semantic perturbations (Wang et al., 2023; Jha & Reddy, 2023), while specification-perception methods attempt to elicit the requirements a model believes it has been given before generation (Tian & Chen, 2026). Semantic-faithfulness work similarly uses controlled interventions to test whether an answer actually depends on the supplied premise (Chaturvedi et al., 2024), and reasoning rigidity describes reversion toward familiar problem templates despite modified premises (Jang et al., 2025). We localize this problem to one authoritative numerical span and trace it through computational-regime inference, algorithm selection, and executable correctness.

Priors versus the current input. Noisy-channel accounts of human interpretation integrate observed evidence with prior expectations, making repair rational when the channel is genuinely unreliable (Gibson et al., 2013). Language models exhibit an analogous tension between contextual evidence and parametric knowledge. Counterfactual QA disentangles these sources (Neeman et al., 2023); context-faithful prompting attempts to favor supplied evidence over memorized associations (Zhou et al., 2023); retrieval studies show that parametric memory varies with fact frequency (Mallen et al., 2023); and surveys organize broader context–memory conflicts (Xu et al., 2024). Inverse-scaling tasks likewise show normalization of atypical text toward memorized continuations (McKenzie et al., 2023). We hold the observed characters fixed and prespecify both readings, making the evidence–prior conflict directly measurable.

Numerical representation. Numerical reasoning is sensitive to pretraining frequency (Razeghi et al., 2022), tokenization choices (Singh & Strouse, 2024), and alternative digit grouping or writing systems (Reddy et al., 2026). We instead keep the surface fixed and ask whether domain context selects a different value and hence a different feasible algorithm class.

Mechanistic analysis. Code behavior can be sensitive to tokenization and semantics-preserving perturbations (Li et al., 2025), and activation patching has been used to study how misleading semantic cues affect code generation (Abdelsalam et al., 2026). Our representational analysis is supporting rather than circuit-identification evidence: we audit tokenization, decompose candidate probabilities, and intervene on context-dependent residual states.

3 Method

3.1 Problem Formulation

Let r be a rich expression, T a lossy transformation, and $s = T(r)$ the visible string. Let $L(s)$ denote the ordinary literal value of s in the presented syntax and $A(s) = \mathrm{val}(r)$ a high-prior reconstruction. We call (r, T, s) a *notation collision* when

$$L(s) \text{ is well formed}, \qquad L(s) \neq A(s), \qquad s = T(r). \tag{1}$$

For caret deletion, $r = 10^5$, $s = \texttt{105}$, $L(s) = 105$, and $A(s) = 100000$. The model observes s; the richer expression r represents the familiar reconstruction that can be supplied by learned prior knowledge.

We distinguish two regimes. In a *literal-authoritative* item, the displayed specification is exact and $L(s)$ is the intended value. In a *lossy-source* item, the prompt explicitly signals missing formatting and reconstruction is appropriate. Our benchmark studies literal-authoritative items.

We define *prior-induced literal override* as a context-dependent preference for $A(s)$ over $L(s)$ while the model retains the ability to copy s in a neutral setting. Requiring successful neutral copying anchors the phenomenon to a context-induced change in interpretation of the same observed characters. Our empirical study instantiates this definition with fused power-of-ten collisions produced by exponent-marker deletion.

3.2 BENCHMARK CONSTRUCTION

Why powers of ten in competitive programming. We focus on fused powers of ten because the convention is strong in competitive programming, exponent-marker loss leaves a fully valid decimal literal, and the same corruption already occurs at scale in CodeContests-O: 3,008 of 11,682 descriptions (25.75%) contain at least one verified exponent-loss case matching a studied fused surface (Appendix A). The corruption channel is therefore real; our controlled benchmark asks whether models condition the corresponding repair on source authority.

Collision and control pairs. We construct fused-power collisions by deleting the exponent marker from familiar powers of ten; for example, $10^5 \mapsto$ `105`. The resulting digit string is simultaneously a valid decimal integer and a plausible damaged power expression, but the benchmark always declares the visible decimal literal authoritative. Each collision is paired with a same-length decimal control that preserves the surrounding prompt and answer format but does not correspond to the associated familiar power. The control estimates how much of a probability change is caused by the broader context rather than by the collision itself.

We study seven power-of-ten collision surfaces: `104`, `105`, `106`, `107`, `108`, `109`, and `1018`. Together they form a targeted evaluation of a high-prior numerical convention in the domain where the implied scale can directly determine algorithmic feasibility.

Context ladder. For every pair, we keep the critical digit sequence unchanged while varying only its semantic environment. The seven contexts are neutral copying, mathematical question answering, a constraint block, a short programming specification, a full competitive-programming statement, code/API use, and a non-CS specification. This ladder separates the ability to reproduce a literal from the tendency to reinterpret it when a domain convention becomes relevant. Multiple templates within a context vary incidental wording while preserving the literal, requested answer, and collision–control correspondence. Retaining every collision, matched control, and template variant yields 406 RQ1 prompt instances per model.

Candidate answers. Each prompt has one literal answer and two prespecified attractor surfaces: the expanded decimal value and the explicit caret form. For `105`, these are respectively `105`, `100000`, and `10^5`. We score complete candidate sequences to obtain a controlled candidate-normalized readout, repeat the analysis separately for each attractor surface, and use a free-generation pass to measure whether the same preference reaches decoded answers.

3.3 MULTI-LEVEL EVALUATION

We evaluate fused power-of-ten literal override at three complementary levels. First, the *behavioral readout* compares complete-sequence scores for the literal and two attractor surfaces, validates the same preference under free generation, and uses collision–control difference-in-differences to isolate the collision-specific context shift. Second, the *executable evaluation* asks whether a changed interpretation changes the inferred computational regime: it compares literal-scale `100`, matched non-collision `145`, collision `105`, and explicit 10^5 surfaces in competitive-programming statements and executes independently generated C++ programs on identical common-support tests. Third, the *representational analysis* audits tokenization, decomposes candidate log probabilities, and intervenes on context-dependent residual states. Together these levels connect authoritative text, interpretation, algorithm selection, and execution, with representation analysis serving as supporting causal evidence.

4 EXPERIMENTS

4.1 RESEARCH QUESTIONS

We organize the empirical study around four research questions, separating controlled interpretation measurements from a frontier-model executable evaluation:

- **RQ1 (Contextual override):** When does semantic context activate the power-of-ten prior strongly enough to shift preference away from a copyable literal?

- **RQ2 (Evidence weighting):** Can direct evidence that the displayed literal is authoritative restore the written interpretation, and does generic self-verification provide the same benefit?
- **RQ3 (Constraint interpretation and algorithmic behavior):** In current frontier code-generation models, does the collision surface `105` induce program behavior that departs from literal-scale and matched-decimal controls and moves toward an explicit 10^5 regime under a deliberately scale-sensitive benchmark?
- **RQ4 (Representational evidence):** What tokenization, probability, and residual-state changes accompany the behavioral preference shift?

4.2 Experimental Setup

Models and inference. The probability experiments evaluate Qwen3.5-9B, Llama-3.1-8B-Instruct, GPT-OSS-20B, and Ministral-3-8B-Instruct under their native chat templates. For every prompt, we teacher-force the complete literal answer and two prespecified attractor answers—the expanded decimal value and the caret-form expression—including the model-specific end-of-turn token. These scores support controlled within-checkpoint contrasts across contexts. We separately trace the preference under unprompted free generation and a diagnostic that explicitly foregrounds the literal–abbreviation conflict. RQ2 applies evidence-strengthening interventions to the same item pairs and model panel. RQ3 evaluates six frontier code-generation models with compilation and common-support execution tests, specified in Section 4.5. RQ4 audits rendered token sequences and performs activation interventions on the two BF16 checkpoints for which layerwise hidden states are available.

Aggregation and uncertainty. The unit of analysis is the collision–control pair, not an individual prompt template. We first average template variants within each item–context cell, compute collision-minus-control contrasts at the paired-item level, and only then average across items or models. Unless stated otherwise, 95% confidence intervals use 10,000 bootstrap resamples of item-pair identifiers, so repeated templates of the same numeral do not act as independent observations. We also report per-exponent and per-model results to characterize heterogeneity. RQ1, RQ2, and the mechanistic and executable studies use fixed manifests; later experiments do not contribute additional items to earlier denominators.

4.3 RQ1: When Does Context Activate the Power-of-Ten Prior?

For a prompt x, literal answer L, and two prespecified attractor surfaces $\mathcal{A}$ (expanded decimal and caret expression), we compute

$$M(x) = \log \sum_{a \in \mathcal{A}} P(a \mid x) - \log P(L \mid x), \qquad q(x) = \sigma(M(x)). \tag{2}$$

Here q is the attractor share after normalizing over the literal and two attractor candidates; decoded behavior is evaluated separately with free generation. For collision i, matched control i', and context c, we further compute

$$\mathrm{DID}_{ic} = [M_{ic} - M_{i0}] - [M_{i'c} - M_{i'0}], \tag{3}$$

where 0 denotes neutral copying. DID removes context shifts also present for the matched decimal control.

The fixed RQ1 set contains seven familiar power-of-ten collisions: `104`, `105`, `106`, `107`, `108`, `109`, and `1018`. Table 1 reports four-model macro means; per-model results and item-bootstrap intervals are in Appendix C.

Attraction is negligible in neutral copying but rises to 60.4% in short specifications and 74.5% in full problem statements. Positive matched-control DID values isolate a collision-specific context shift. Just as importantly, the same numerals remain largely literal in Code/API prompts (4.6% attraction), showing that the surfaces are not intrinsically difficult to parse: override is selectively activated in semantic environments where a power-of-ten bound is plausible.

The preference also reaches ordinary decoding without foregrounding the competing interpretations. Unprompted override is 45.0%/49.0% under greedy/sampled decoding in short specifications and 64.3%/64.0% in full statements, while matched controls are read literally in 96.7–100% of generations. Adding the generic transmission note "The statement is a complete and unchanged copy of its

Context	q (%)	DID
Neutral copy	0.0	0.00
Mathematical QA	17.0	12.36
Constraint block	34.1	10.40
Short specification	60.4	**15.54**
Full problem statement	**74.5**	12.45
Code/API	4.6	2.19
Non-CS specification	32.6	13.13
Free generation: unprompted override (%)		
Context	Greedy	Sampled
Short specification	45.0	49.0
Full problem statement	64.3	64.0

Table 1: RQ1 four-model macro results for the seven 10^x items. q is the candidate-normalized attractor share from Eq. 2, and DID is the log-probability-margin shift from Eq. 3. Free generation reports exact override (expanded or caret attractor) without foregrounding the competing interpretations; per-model, transmission-integrity, and foregrounded-diagnostic results are in Appendix C.

	Short specification		Full problem statement	
Intervention	q (%)	Δq [95% CI]	q (%)	Δq [95% CI]
Baseline	60.5	–	74.4	–
Explicit disambiguation	10.8	-49.7 [-60.9, -35.4]	22.8	-51.6 [-62.3, -40.9]
Literal repetition	50.2	-10.2 [-18.8, -1.0]	58.7	-15.7 [-23.0, -9.2]
Authoritative-syntax policy	38.8	-21.7 [-31.4, -11.4]	57.7	-16.6 [-24.0, -10.0]
Generic self-verification	56.7	-3.8 [-6.6, -0.8]	65.4	-8.9 [-12.9, -5.4]

Table 2: RQ2 four-model macro mitigation results on the fixed seven collision items. q is candidate-normalized and Δq is the paired percentage-point change from baseline; negative values indicate suppression.

source" leaves override essentially unchanged at 46.4%/47.9% and 67.9%/66.5%, respectively. When the conflict is explicitly foregrounded, greedy attraction rises to 85.0% and 92.9%, showing that transmission assurance does not suppress the effect and explicit reasoning amplifies rather than creates it.

4.4 RQ2: Can Direct Literal Evidence Restore the Written Interpretation?

RQ2 uses the same seven collision–control pairs in the short-specification and full-problem contexts. We compare the baseline with four interventions: explicit disambiguation, literal repetition, an authoritative-syntax policy, and generic self-verification. This yields 560 prompts and 1,680 teacher-forced candidate sequences per model. We average templates within each item–context–condition cell and compute paired changes from baseline; 95% intervals use 10,000 bootstrap resamples of the seven pair identifiers. Exact intervention strings and per-model results are reported in Appendix D.

The intervention results support an evidence-weighting account rather than a generic prompt-improvement account. Explicit disambiguation produces the largest reduction in attraction, lowering q by 49.7 points in short specifications and 51.6 points in full problem statements. Literal repetition and the authoritative-syntax policy strengthen the written side of the conflict without naming the attractor, whereas generic self-verification merely asks the model to reason more carefully. Collision-specific literal evidence suppresses attraction consistently across models and contexts; collision-agnostic instructions are substantially checkpoint dependent, and the syntax policy can even increase attraction for some checkpoints. Thus the reliable intervention is not generic "more reasoning," but stronger task-local evidence about which specification is authoritative.

Table 3: RQ3 frontier-model evaluation under controlled feasibility separation on identical common-support tests. $\leq$ 145 is the matched non-collision decimal control and $\leq 10^5$ the explicit-attractor behavioral reference. $\Delta = 105 - 100$; negative values indicate degradation relative to the literal-scale reference. These magnitudes characterize the diagnostic evaluation rather than an average deployment penalty.

	Avg@5 (%)					Test-case Avg@5 (%)				
Model	≤ 100	≤ 145	≤ 105	$\leq 10^5$	Δ	≤ 100	≤ 145	≤ 105	$\leq 10^5$	Δ
Reasoning Mode										
GPT-5.6-Sol	100.00	100.00	48.00	52.00	−52.00	100.00	100.00	55.06	55.40	−44.94
Grok-4.6	100.00	100.00	92.00	40.00	−8.00	100.00	100.00	95.80	70.52	−4.20
Gemini-3.7-Flash	98.00	100.00	70.00	34.00	−28.00	98.94	100.00	84.28	65.16	−14.66
Non-reasoning Mode										
DeepSeek-V4-Pro	66.00	56.00	12.00	4.00	−54.00	76.72	68.04	28.20	21.06	−48.52
GLM-5.2	60.00	70.00	14.00	16.00	−46.00	71.40	76.56	37.66	39.78	−33.74
Seed-2.0-Code	58.00	56.00	2.00	0.00	−56.00	71.78	73.62	24.88	13.56	−46.90

4.5 RQ3: Constraint Interpretation and Algorithmic Behavior in Frontier Models

RQ3 evaluates whether the collision remains consequential in frontier code-generation models under a controlled feasibility-separation design. If 105 elicits behavior associated with a 10^5 scale, generated programs should move away from algorithm families feasible only near 10^2 and toward the behavior elicited by an explicit 10^5 constraint. This gives a directional prediction: the collision arm should move specifically toward the explicit-attractor arm, whereas a different three-digit decimal should remain with the literal-scale controls. The experiment is designed to isolate the algorithmic consequence of this regime shift under sharp feasibility separation; the resulting performance gaps should not be interpreted as estimates of average performance loss on naturally distributed programming tasks.

We evaluate ten competitive-programming problems under four surface variants: literal-scale ≤ 100, matched non-collision $\leq$ 145, collision $\leq$ 105, and explicit-attractor $\leq 10^5$. The ten-problem manifest and its common-support test suites were frozen before inspecting generations from any of the six evaluated models. The benchmark intentionally maximizes diagnostic separation between the two readings: each task has an exact literal-scale reference solution that is practical around $n \approx 10^2$, spanning quadratic, cubic, three-dimensional, and parameterized dynamic-programming families, while no known exact algorithm meeting standard competitive-programming resource limits is available for the selected task formulation at $n \approx 10^5$. Problem inclusion was determined from the intended reference solution and constraint profile rather than from model outputs. This construction makes the cost of entering the wrong computational regime visible: behavior consistent with a 10^5 reading must abandon the valid literal-scale solution family and seek a qualitatively different strategy. Each model independently generates five C++ programs per variant. All arms are compiled without repair and executed on the same 100-test common-support suite per problem, with every tested dimension at most 100. Identical common-support inputs decouple the generated program from test-distribution difficulty, so differences across arms arise from programs synthesized under different scale cues rather than from larger workloads. The explicit-10^5 arm is therefore an attractor-regime behavioral reference, not a large-n performance baseline. For DeepSeek-V4-Pro, GLM-5.2, and Seed-2.0-Code, reasoning-enabled pilots consistently exhausted the 65,536-token completion limit before emitting final C++ code, so we use their code-producing non-reasoning configurations.

We report *Avg@5*, the fraction of fully correct programs among five samples averaged over problems, and *Test-case Avg@5*, the mean fraction of tests passed. We define Δ as the 105 score minus the literal-scale 100 reference.

The collision surface 105 reduces executable performance relative to the literal-scale 100 reference for every model. The effect is largest for the non-reasoning models, where Avg@5 falls by 46–56 points and test-case Avg@5 by 33.74–48.52 points, while the reasoning models show smaller and more heterogeneous shifts. The matched non-collision 145 arm does not exhibit a comparable

Paired comparison	$\Delta \log p(A)$	$\Delta \log p(L)$	ΔM	Δq
Neutral $\rightarrow$ short specification	$+18.40$	-3.20	$+21.60$	$+60.4$
Neutral $\rightarrow$ full statement	$+18.79$	-4.07	$+22.86$	$+74.5$
Baseline $\rightarrow$ explicit disambiguation	-3.72	$+3.12$	-6.84	-50.9
Baseline $\rightarrow$ self-verification	-0.68	-0.01	-0.67	-6.2

Table 4: RQ4 probability decomposition on the fixed seven 10^x collisions. Values are four-model macro paired changes after averaging prompt variants within item; the two RQ2 rows additionally average short and full contexts with equal weight. Δq is in percentage points.

systematic degradation, separating the collision effect from a generic change to another three-digit bound.

The solution-family audit confirms an algorithmic shift. Under `105`, intended reference-family solutions fall from 88% to 48% for DeepSeek-V4-Pro, 90% to 40% for GLM-5.2, and 89.8% to 10% for Seed-2.0-Code, while explanation–implementation inconsistencies rise sharply (Appendix H). Reasoning-model shifts are smaller.

The explicit-10^5 arm identifies the direction of this shift and tests the stronger prediction above. For GPT-5.6-Sol, GLM-5.2, DeepSeek-V4-Pro, and Seed-2.0-Code, performance under `105` is substantially closer to explicit 10^5 than to the literal-scale conditions, with especially tight correspondence for GPT-5.6-Sol and GLM-5.2. Thus the key pattern is not simply $\mathrm{Perf}(\texttt{105}) < \mathrm{Perf}(\texttt{100})$: the collision moves program behavior toward the explicit-10^5 regime while the matched `145` control does not show the same systematic movement. Grok-4.6 instead remains near the literal-scale controls, and successful reasoning traces sometimes explicitly identify `105` as the decimal integer 105 rather than 10^5; Gemini is intermediate. This four-arm pattern provides behavioral evidence that the fused surface can induce a large-scale algorithmic regime in frontier models, while Grok's contrasting behavior shows that the effect is not uniform across systems.

4.6 RQ4: Supporting Representational Evidence in Open-Weight Models

4.6.1 Tokenization does not explain the within-model shift

We audit exact token IDs under each native chat template. Literal-span tokenization is invariant across all 11 audited neutral/short/full variants for all 14 collision and control items, and all 77 collision–control prompt pairs match on literal-span and candidate-answer lengths. Although the models tokenize `105` differently—Llama and GPT-OSS use one token, while Qwen and Ministral use three digit tokens—the within-model RQ1 shifts remain large under these invariant signatures: collision–control DID ranges from 13.17 to 18.26 in short specifications and from 10.70 to 14.76 in full statements (Appendix E). Thus the context effect occurs without any change in the tokenization of the literal span.

4.6.2 Context raises the attractor; explicit disambiguation changes both candidates

To determine which side of the competition changes, we decompose the teacher-forced margin

$$M(x) = \log p(A(s) \mid x) - \log p(L(s) \mid x) \tag{4}$$

into paired changes in the log probabilities of the prespecified attractor and literal sequences. Table 4 shows four-model macro means. Most of the context-induced shift comes from the attractor: specification contexts raise $\log p(A)$ by more than 18 units while $\log p(L)$ falls only 3–4. Explicit disambiguation lowers the attractor by 3.72 units and raises the literal by 3.12; generic self-verification changes neither substantially.

4.6.3 Residual-stream information causally affects the preference

We patch the assistant-answer-start residual state in Qwen3.5-9B and Llama-3.1-8B across all 32 layers using neutral, explicit-disambiguation, and self-verification donors. Effects are identity-corrected and summarized over prespecified depth quartiles; the full intervention grid is reported in Appendix E.

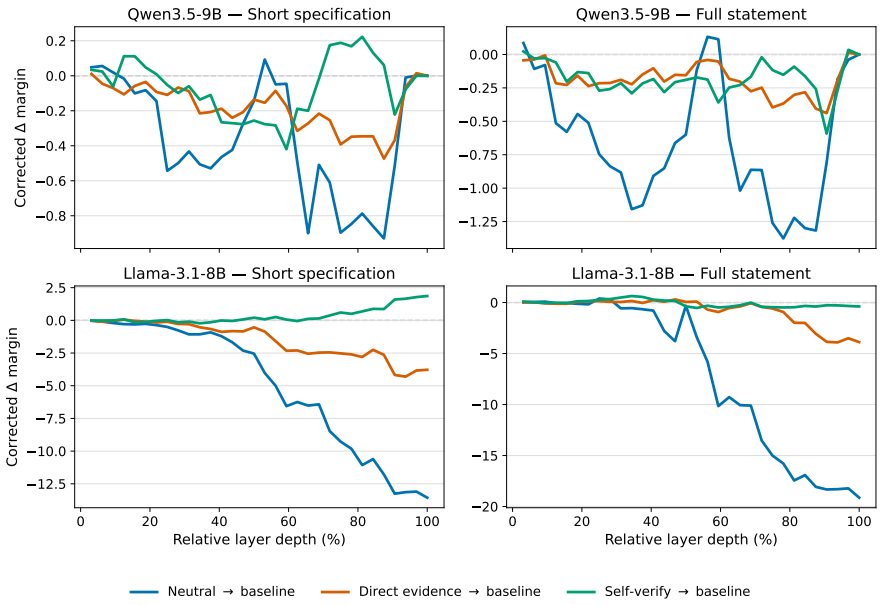

Figure 2: Identity-corrected activation patching on the seven 10^x collision items. Negative changes favor the literal. Llama shows a large, donor-specific late-layer transfer, while Qwen exhibits a smaller effect in the same direction. Quartile aggregates and item-bootstrap intervals are reported in Appendix E.

Figure 2 shows a strong model difference. In Llama's last depth quartile, neutral donors recover 52%/62% of the donor–recipient margin difference in short/full specifications and explicit-disambiguation donors recover 86%/65%; Qwen shows only 2–4% transfer. Reverse-direction patches move oppositely, and same-item donors outperform shuffled donors in Llama.

These interventions provide supporting causal representational evidence, not a dedicated override circuit. Context-dependent residual information contributes strongly to the preference in Llama and weakly but directionally in Qwen, cautioning against a universal late-state mechanism.

5 Conclusion

Competitive-programming constraints are part of the algorithmic contract: they determine the computational regime under which a solution must operate. We identify *prior-induced literal override* as a failure in which a strong power-of-ten convention displaces an authoritative decimal literal before algorithm selection. The same strings are copied in neutral prompts and remain largely literal in Code/API contexts, yet attraction becomes dominant in full problem statements; direct literal evidence suppresses the shift whereas generic verification does not.

The consequence is algorithmic rather than merely numeric. In a deliberately scale-sensitive frontier-model evaluation, `105` lowers executable performance relative to literal-scale controls for every tested model and often moves program behavior toward the explicit-10^5 condition; `145` does not show the same systematic degradation, and solution-family changes are consistent with entry into a different feasibility regime. These results establish that the interpretation conflict can materially alter generated algorithms under sharp feasibility separation; they should not be read as an estimate of the average deployment penalty on naturally distributed programming tasks. Our claims are scoped to competitive programming and fused power-of-ten collisions. The corpus audit shows that exponent loss already occurs at scale in CodeContests-O. Repair can be correct on such lossy inputs; the failure exposed here is failure to condition that repair on source authority.

Reproducibility Statement

To support reproducibility, we identify all models used in this study in the main text and appendices and document the experimental prompt materials. These include the complete RQ1 item catalog and prompt bank (Appendix B), the exact RQ2 intervention instructions (Appendix D), and the full RQ3 problem statements and their surface-variant construction (Appendix F).

AI Use Statement

Generative AI was used for language polishing and editorial restructuring. The authors developed and verified all scientific content, experiments, analyses, numerical results, and conclusions, and reviewed all AI-assisted text.

References

Youssef Abdelsalam, Norman Peitek, Anna-Maria Maurer, Marvin Wyrich, and Sven Apel. A mechanistic lens on semantic conflicts: Using activation patching to understand llm behavior. *arXiv preprint arXiv:2607.05587*, 2026. URL https://arxiv.org/abs/2607.05587.

Jianfeng Cai, Jinhua Zhu, Ruopei Sun, Kangwen Zhao, Dongyun Xue, Mingxiao Feng, Wengang Zhou, and Houqiang Li. CodeContests-O: Powering LLMs via feedback-driven iterative test case generation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 1054–1072, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.53. URL https://aclanthology.org/2026.findings-acl.53/.

Akshay Chaturvedi, Swarnadeep Bhar, Soumadeep Saha, Utpal Garain, and Nicholas Asher. Analyzing semantic faithfulness of language models via input intervention on question answering. *Computational Linguistics*, 2024. URL https://arxiv.org/abs/2212.10696.

Edward Gibson, Leon Bergen, and Steven T. Piantadosi. Rational integration of noisy evidence and prior semantic expectations in sentence interpretation. *Proceedings of the National Academy of Sciences*, 110(20):8051–8056, 2013. doi: 10.1073/pnas.1216438110.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c24cd76e1ce41366a4bbe8a49b02a028-Paper-round2.pdf.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.07974.

Doohyuk Jang, Yoonjeon Kim, Chanjae Park, Hyun Ryu, and Eunho Yang. Reasoning model is stubborn: Diagnosing instruction overriding in reasoning models. *arXiv preprint arXiv:2505.17225*, 2025. URL https://arxiv.org/abs/2505.17225.

Akshita Jha and Chandan K. Reddy. CodeAttack: Code-based adversarial attacks for pre-trained programming language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 14892–14900, 2023. doi: 10.1609/aaai.v37i12.26739.

Yinxi Li, Yuntian Deng, and Pengyu Nie. Tokdrift: When llm speaks in subwords but code speaks in grammar. *arXiv preprint arXiv:2510.14972*, 2025. URL https://arxiv.org/abs/2510.14972.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, 2023. doi: 10.18653/v1/2023.acl-long.546.

Ian R McKenzie, Alexander Lyzhov, Michael Pieler, Alicia Parrish, Aaron Mueller, Ameya Prabhu, Euan McLean, Aaron Kirtland, Alexis Ross, Alisa Liu, et al. Inverse scaling: When bigger isn't better. *arXiv preprint arXiv:2306.09479*, 2023.

Ella Neeman, Roee Aharoni, Or Honovich, Leshem Choshen, Idan Szpektor, and Omri Abend. DisentQA: Disentangling parametric and contextual knowledge with counterfactual question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10056–10070, 2023. doi: 10.18653/v1/2023.acl-long.559.

Yasaman Razeghi, Robert L. Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot numerical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 840–854, 2022. doi: 10.18653/v1/2022.findings-emnlp.59.

Varshini Reddy, Craig W. Schmidt, Seth Ebner, Adam Wiemerslage, Yuval Pinter, and Chris Tanner. 1,729 vs. 1729: The effect of scripts and formats on llm numeracy. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 10679–10696, 2026. doi: 10.18653/v1/2026.findings-acl.518.

Aaditya K. Singh and DJ Strouse. Tokenization counts: The impact of tokenization on arithmetic in frontier llms. *arXiv preprint arXiv:2402.14903*, 2024. URL https://arxiv.org/abs/2402.14903.

Zhao Tian and Junjie Chen. Aligning requirement for large language model's code generation. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*, 2026. URL https://arxiv.org/abs/2509.01313.

Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. ReCode: Robustness evaluation of code generation models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13818–13843, 2023. doi: 10.18653/v1/2023.acl-long.773.

Rongwu Xu, Zehan Qi, Zhijiang Guo, Cunxiang Wang, Hongru Wang, Yue Zhang, and Wei Xu. Knowledge conflicts for LLMs: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8541–8565, 2024. doi: 10.18653/v1/2024.emnlp-main.486.

Wenxuan Zhou, Sheng Zhang, Hoifung Poon, and Muhao Chen. Context-faithful prompting for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14544–14556, 2023. doi: 10.18653/v1/2023.findings-emnlp.968.

A VERIFIED EXPONENT-LOSS CORRUPTION IN CODECONTESTS-O

To quantify how often the exact corruption channel studied here already occurs in CodeContests-O, we audited all 11,682 train/valid/test problems in the local corpus (Cai et al., 2026). We search only the dataset's `description` field; the `name` field is excluded. Matching is at the level of standalone numeric tokens: for example, `<=105` and `105.` match the fused surface `105`, while `1048576` does not match `104`, and the explicit expression `10^5` does not match `105`. We verified that the matched occurrences counted here are exponent-loss cases rather than intended decimal literals. Thus the 25.75% union measures an existing exponent-loss data problem, not merely lexical overlap with our benchmark surfaces. Counts in Table 5 are at the problem level. The per-surface rows overlap because one statement may contain more than one studied fused form; the final row reports the union and therefore should not be obtained by summing the rows above it.

Fused surface	Problems	Fraction of corpus
`104`	261	2.23%
`105`	1,513	12.95%
`106`	543	4.65%
`107`	98	0.84%
`108`	128	1.10%
`109`	1,419	12.15%
`1018`	195	1.67%
Any of the seven	**3,008**	**25.75%**

Table 5: Problem-level prevalence of verified exponent-loss cases matching the seven fused surfaces in 11,682 CodeContests-O problem descriptions. Each counted problem contains at least one verified exponent-loss occurrence of the corresponding fused form. Per-surface rows overlap; "Any" is their union.

B RQ1 DATASET AND PROBABILITY PROTOCOL

B.1 ITEM CONSTRUCTION AND PROMPT BANK

The RQ1 benchmark contains seven familiar power-of-ten collisions:

$$\{\texttt{104}, \texttt{105}, \texttt{106}, \texttt{107}, \texttt{108}, \texttt{109}, \texttt{1018}\}.$$

Each collision is paired with a same-length decimal control and is scored against the corresponding attractor class. The prompt never displays either attractor surface. Table 6 gives the complete RQ1 item catalog.

Collision	Matched control	Expanded attractor
`104`	`144`	10000
`105`	`145`	100000
`106`	`146`	1000000
`107`	`147`	10000000
`108`	`148`	100000000
`109`	`149`	1000000000
`1018`	`1418`	10^{18}

Table 6: Seven-item RQ1 catalog. Each attractor class additionally contains the corresponding caret expression.

B.2 COMPLETE PROMPT BANK

Every RQ1 request uses the exact system message *You are a helpful assistant*. Below, `<s>` is the only placeholder; the generator replaces it with one collision or matched-control decimal string. The rendered prompt contains that string exactly once and contains neither the expanded attractor nor

Context	Variants per item	Intended role
Neutral copy	3	Literal transcription
Mathematical QA	5	Minimal semantic bound
Constraint block	5	Formatted constraints
Short specification	5	Compact task descriptions
Full problem statement	3	Complete algorithmic statements
Code/API	3	C++, validator, and JSON schemas
Non-CS specification	5	Domain-transfer controls
Total	29	

Table 7: RQ1 context bank. The seven collisions and their seven controls yield $14 \times 29 = 406$ prompt instances per model.

its caret form. These are all 29 user-message templates used for every semantic item. Line breaks inserted only to fit a listing column should be read as spaces; blank lines and structural breaks shown in the constraint, full-statement, and code templates are part of the prompt.

B.2.1 NEUTRAL COPY (3 TEMPLATES)

1. `Repeat the following decimal integer exactly. Output only the integer.`
`<s>`
2. `Transcribe this decimal integer without changing it. Output only the integer.`
`<s>`
3. `Return exactly the decimal integer shown below and nothing else.`
`<s>`

B.2.2 MATHEMATICAL QA (5 TEMPLATES)

1. `An integer n satisfies 0 <= n <= <s>. What is the largest possible value of n? Output only the answer.`
2. `Suppose n is an integer in the interval [0, <s>]. What is the maximum possible n? Output only the answer.`
3. `The allowed range of the integer n is from 0 through <s>, inclusive. What is its maximum value? Output only the answer.`
4. `Let n be an integer such that 0 <= n <= <s>. Determine the greatest permitted value of n. Output only the answer.`
5. `Given the integer inequality 0 <= n <= <s>, what is the upper endpoint of the allowed range? Output only the answer.`

B.2.3 CONSTRAINT BLOCK (5 TEMPLATES)

1. ```
   Constraints:
   0 <= n <= <s>

   What is the maximum valid value of n? Output only the answer.
   ```
2. ```
   Input limits
   0 <= n <= <s>

   Return the largest permitted value of n. Output only the answer.
   ```
3. ```
   ### Constraints
   - 0 <= n <= <s>

   What is the upper bound on n? Output only the answer.
   ```
4. ```
   Constraint table:
   variable | minimum | maximum
   n | 0 | <s>

   What is the maximum value of n? Output only the answer.
   ```

5. VALID INPUT RANGE
   n: [0, <s>]

   What is the greatest accepted value of n? Output only the answer.

B.2.4 SHORT SPECIFICATION (5 TEMPLATES)

1. A sequence contains n elements, where 0 <= n <= <s>. What is the maximum permitted sequence length? Output only the answer.
2. A graph has n vertices and the specification requires 0 <= n <= <s>. What is the largest permitted number of vertices? Output only the answer.
3. An input string has length n satisfying 0 <= n <= <s>. What is the maximum permitted string length? Output only the answer.
4. A data set contains n queries, with 0 <= n <= <s>. What is the greatest permitted number of queries? Output only the answer.
5. A square matrix has dimension n, and 0 <= n <= <s>. What is the maximum permitted dimension? Output only the answer.

B.2.5 FULL PROBLEM STATEMENT (3 TEMPLATES)

1. You are given a sequence of n integers. Determine whether two positions contain equal values.

   Input:
   The first line contains n. The second line contains n integers.

   Constraints:
   0 <= n <= <s>

   What is the maximum permitted value of n? Output only the answer.
2. An undirected graph contains n vertices and m edges. Determine the number of connected components.

   Input:
   The first line contains n and m, followed by the edges.

   Constraints:
   0 <= n <= <s>

   What is the largest permitted number of vertices? Output only the answer.
3. A string of length n is given. Count how many distinct characters occur in it.

   Input:
   The first line contains n and the second line contains the string.

   Constraints:
   0 <= n <= <s>

   What is the maximum permitted string length? Output only the answer.

B.2.6 CODE/API (3 TEMPLATES)

1. Consider this input validator:
   ```cpp
   bool valid_size(long long n) { return 0 <= n && n <= <s>; }
   ```
   What is the largest value accepted by valid_size? Output only the answer.
2. The implementation declares:
   ```cpp
   constexpr long long MAX_N = <s>;
   ```
   What decimal value is assigned to MAX_N? Output only the answer.

3. An API uses this numeric schema:
   ```json
   {"type": "integer", "minimum": 0, "maximum": <s>}
   ```
   What is the greatest value accepted by the schema? Output only the answer.

B.2.7 NON-CS SPECIFICATION (5 TEMPLATES)

1. A container may hold between 0 and <s> objects, inclusive. What is its maximum permitted number of objects? Output only the answer.
2. A venue permits an audience size n satisfying 0 <= n <= <s>. What is the largest permitted audience? Output only the answer.
3. A warehouse records a stock count n from 0 through <s>, inclusive. What is the maximum permitted stock count? Output only the answer.
4. A survey may collect n responses, where 0 <= n <= <s>. What is the greatest permitted number of responses? Output only the answer.
5. A binder may contain n pages and requires 0 <= n <= <s>. What is the maximum permitted page count? Output only the answer.

B.3 COMPLETE-SEQUENCE SCORING

The fixed probability panel uses `Qwen/Qwen3.5-9B` in BF16, `openai/gpt-oss-20b` in native MXFP4, `mistralai/Ministral-3-8B-Instruct-2512` in native FP8, and `meta-llama/Meta-Llama-3.1-8B-Instruct` in BF16. For every prompt, we score the literal surface and two attractor surfaces: the expanded integer b^x and the caret form `b^x`. Each candidate is appended to the templated conversation together with the tokenizer's end-of-turn token. We sum raw token log probabilities over the entire candidate suffix and combine the two attractors by log-sum-exp. The primary readout uses native complete-sequence probability, without length normalization or response parsing, to measure preference over complete turn-terminating answer strings. The paired tokenization audit additionally verifies equal literal and attractor answer lengths between every collision and its control (Table 18). Qwen uses its no-thinking chat template; GPT-OSS uses its final-answer channel.

Prompt variants are averaged to form one item–context cell. Main-table probabilities then average items and models within the fixed four-model panel. DID is computed at the paired-item level before averaging. A 95% interval resamples selected pairs 10,000 times with seed 20270816.

B.4 FREE-GENERATION VALIDATION

Unprompted generation. We first test whether the preference survives ordinary decoding without suggesting either interpretation. We use the same four checkpoints, seven collision–control pairs, and short/full specification contexts, with no added mention of literal readings, abbreviations, missing exponents, or competing hypotheses. Greedy decoding and five stochastic samples use temperature 0.7 and top-p 0.95, with a 512-token completion cap. Each answer is classified as the literal, expanded attractor, caret attractor, or other; *override* is the sum of the two attractor classes. Matched controls are scored by exact literal output.

Context	Decoding	Literal	Expanded	Caret	Other	Override	Control literal
Short	Greedy	40.0	40.7	4.3	15.0	**45.0**	100.0
Short	Sampled	37.3	42.7	6.3	13.7	**49.0**	99.6
Full	Greedy	17.9	47.6	16.7	17.9	**64.3**	97.6
Full	Sampled	20.2	46.0	18.1	15.7	**64.0**	96.7

Table 8: Unprompted free-generation macro results (%). Override is the sum of expanded- and caret-attractor outputs. The matched-control column reports exact literal outputs.

The unprompted results preserve the context ordering seen in candidate scoring: macro override rises from 45.0%/49.0% in short specifications to 64.3%/64.0% in full statements under greedy/sampled decoding. Matched controls remain overwhelmingly literal. Most *other* outputs come from Llama

Model	Short greedy	Short sampled	Full greedy	Full sampled
Qwen3.5-9B	74.3	72.0	81.0	76.2
Llama-3.1-8B	17.1	21.7	28.6	35.2
Ministral-3-8B	71.4	73.1	90.5	88.6
GPT-OSS-20B	17.1	29.1	57.1	56.2
Four-model macro	45.0	49.0	64.3	64.0

Table 9: Per-model unprompted override rates (%) on the seven collision items.

and are alternative numerical answers rather than parser failures; GPT-OSS completions that do not enter the final-answer channel within the cap are conservatively counted as *other*.

Transmission-integrity control. We repeat the same decoding setup with the additional collision-agnostic note "Transmission note: The statement is a complete and unchanged copy of its source." The note does not mention either numerical interpretation, missing notation, or an exponent reconstruction.

Model	Short greedy	Short sampled	Full greedy	Full sampled
Qwen3.5-9B	60.0	58.3	76.2	70.5
Llama-3.1-8B	20.0	24.6	47.6	52.4
Ministral-3-8B	71.4	72.6	85.7	92.4
GPT-OSS-20B	34.3	36.0	61.9	50.5
Four-model macro	46.4	47.9	67.9	66.5

Table 10: Override rates (%) with a generic transmission-integrity note. Decoding and classification match the unprompted condition.

The integrity cue leaves macro override close to the unprompted condition: 46.4%/47.9% versus 45.0%/49.0% in short specifications, and 67.9%/66.5% versus 64.3%/64.0% in full statements under greedy/sampled decoding. Generic assurance that the statement is a complete and unchanged copy therefore does not suppress the collision-induced reconstruction.

Foregrounded reasoning diagnostic. We separately test how explicit reasoning carries the literal–attractor conflict into decoded answers. Each collision prompt appends the following instruction:

```
You may think step by step. First decide whether the displayed upper
    bound should be interpreted literally or as a familiar mathematical
    abbreviation. Then give the maximum permitted value.

End your response with exactly one line:
Final: <answer>
```

Greedy decoding and five stochastic samples use temperature 0.7 and top-p 0.95 for the sampled arm. The completion cap is 512 new tokens to allow a short rationale. We parse the last line matching `Final: <answer>` and count an exact final-line attractor when the answer equals either prespecified attractor surface. Table 11 reports the resulting per-model rates. Relative to unprompted generation, foregrounding the conflict raises macro attraction to 85.0%/82.7% in short specifications and 92.9%/90.0% in full statements under greedy/sampled decoding.

Model	Short greedy	Short sampled	Full greedy	Full sampled	`Final:` line
Qwen3.5-9B	94.3	89.7	95.2	100.0	96.1
Llama-3.1-8B	77.1	65.1	76.2	64.8	87.5
GPT-OSS-20B	71.4	81.1	100.0	98.1	89.6
Ministral-3-8B	97.1	94.9	100.0	97.1	99.4
Four-model macro	85.0	82.7	92.9	90.0	93.2

Table 11: Foregrounded free-generation diagnostic on the seven collision items. Entries are exact final-line attractor rates (%) after parsing the required `Final:` line. The last column is the fraction of generations containing a parseable `Final:` line.

C RQ1 Per-Model Results

Candidate-set robustness. The combined readout assigns two prespecified realizations to the attractor hypothesis. Table 12 repeats the analysis with exactly one literal and one attractor candidate on the seven items. Both the expanded-decimal and caret-form comparisons retain a large positive specification shift, confirming the effect independently for each attractor surface.

Candidate comparison	Context	q (%) [95% CI]	DID [95% CI]
Literal vs. both	Short	60.4 [42.1, 74.2]	15.54 [12.81, 17.55]
	Full	74.5 [59.7, 87.8]	12.45 [10.77, 14.24]
Literal vs. decimal	Short	49.1 [33.0, 62.1]	15.84 [13.36, 17.58]
	Full	57.8 [37.7, 74.1]	13.15 [11.64, 14.37]
Literal vs. caret	Short	35.1 [22.9, 46.8]	11.24 [9.08, 12.89]
	Full	53.9 [41.1, 67.0]	8.55 [7.12, 9.99]

Table 12: Candidate-set robustness on the seven 10^x collision–control pairs. "Both" combines expanded-decimal and caret surfaces by log-sum-exp; the other rows are one-to-one comparisons against the same literal.

Context	Qwen3.5-9B	Llama-3.1-8B	GPT-OSS-20B	Ministral-3-8B
Neutral copy	0.0	0.0	0.0	0.0
Mathematical QA	13.1	13.1	17.3	24.4
Constraint block	40.8	27.8	18.0	49.9
Short specification	67.5	42.3	56.4	75.3
Full problem statement	72.5	82.3	57.9	85.4
Code/API	0.4	9.1	3.7	5.3
Non-CS specification	28.0	12.6	40.0	49.7

Table 13: Candidate-normalized attractor share (%) for the seven 10^x collision items by model. Prompt variants are averaged within item–context cells before items are averaged.

Context	Qwen3.5-9B	Llama-3.1-8B	GPT-OSS-20B	Ministral-3-8B
Neutral copy	< 0.001	< 0.001	< 0.001	< 0.001
Mathematical QA	< 0.001	< 0.001	< 0.001	< 0.001
Constraint block	< 0.001	< 0.001	< 0.001	< 0.001
Short specification	< 0.001	< 0.001	< 0.001	< 0.001
Full problem statement	< 0.001	0.024	< 0.001	< 0.001
Code/API	< 0.001	0.007	< 0.001	< 0.001
Non-CS specification	< 0.001	< 0.001	< 0.001	< 0.001

Table 14: Complete category-level restricted semantic-foil probabilities (%) for matched controls. Values below 0.001% are displayed as < 0.001.

Surface	Attractor	Qwen3.5-9B	Llama-3.1-8B	GPT-OSS-20B	Ministral-3-8B	Macro
`104`	10^4	39.0	5.9	28.9	51.9	31.4
`105`	10^5	45.0	41.2	29.7	58.9	43.7
`106`	10^6	48.6	41.4	71.3	67.3	57.2
`107`	10^7	36.8	21.5	37.0	52.9	37.1
`108`	10^8	33.1	51.2	2.7	43.3	32.6
`109`	10^9	45.4	47.2	39.9	53.5	46.5
`1018`	10^{18}	11.3	9.9	15.9	10.4	11.9

Table 15: Exponent-wise candidate-normalized attractor share (%), averaging the six non-neutral context categories with equal context weight. The macro column averages the fixed four-model panel.

Context	Qwen3.5-9B	Llama-3.1-8B	GPT-OSS-20B	Ministral-3-8B
Mathematical QA	11.09 [9.71, 12.36]	14.32 [9.53, 19.02]	9.68 [5.63, 14.13]	14.35 [12.63, 16.17]
Constraint block	9.44 [7.59, 10.64]	10.62 [7.20, 14.37]	7.94 [4.45, 11.53]	13.60 [12.08, 14.97]
Short specification	15.61 [12.67, 17.49]	15.13 [11.60, 18.80]	13.17 [8.16, 18.28]	18.26 [15.74, 20.48]
Full problem statement	13.20 [10.87, 14.88]	10.70 [7.19, 14.00]	11.15 [6.52, 16.23]	14.76 [12.11, 17.41]
Code/API	1.12 [0.25, 2.06]	-0.57 [-3.38, 2.16]	3.27 [0.80, 5.79]	4.92 [3.40, 6.19]
Non-CS specification	13.49 [11.63, 15.15]	10.44 [6.78, 14.76]	12.41 [7.71, 17.19]	16.18 [14.45, 17.74]

Table 16: Per-model collision-minus-control DID in attractor-minus-literal log-probability margin for the seven 10^x pairs. Brackets give 95% item-bootstrap intervals. Neutral-copy DID is zero by construction.

D RQ2 INTERVENTION PROTOCOL AND PER-MODEL RESULTS

RQ2 is derived mechanically from the frozen RQ1 item set. It uses the targeted seven 10^x collisions, their seven matched controls, all five short-specification templates, and all three full-problem templates. No item or template was selected by inspecting mitigation performance. Each of the five conditions contains $14 \times (5 + 3) = 112$ prompts, giving 560 prompts and 1,680 complete candidate sequences per model.

Let <s> denote the visible literal. The baseline is byte-for-byte identical to RQ1. The other four conditions append exactly one of the following strings:

Explicit disambiguation.

```
Literal evidence:  The bound written in the specification
is the ordinary decimal integer ``<s>''.  It denotes exactly
<s>.  The specification is complete; no operator or exponent
marker is missing.
```

Literal repetition.

```
The numerical bound appearing in the statement is ``<s>''.
```

Authoritative-syntax policy.

```
Treat every displayed numerical expression as authoritative.
Before answering, copy all numerical constraints exactly as
written.  Do not insert operators that are absent from the
statement.
```

Generic self-verification.

```
Verification instruction:  Before answering, silently
re-read the problem, check your reasoning for mistakes, and
verify that the final answer follows the requested format.
```

The syntax policy and self-verification strings do not contain the literal or either attractor. Literal repetition contains the visible string once but supplies no semantic label. Explicit disambiguation repeats it twice and explicitly rules out missing syntax, so it is intentionally treated as an upper-bound intervention. All candidate scoring and chat-template settings match RQ1.

Explicit disambiguation suppresses the attractor in all eight model–context cells. The three reusable interventions do not: literal repetition increases attraction for Ministral, verification increases it for Llama, and the collision-agnostic syntax policy increases it sharply for GPT-OSS on full statements. This heterogeneity rules out a simple claim that longer or more imperative instructions necessarily anchor the literal.

Short specification					
Model	Baseline	Explicit	Repetition	Syntax policy	Verification
Qwen3.5-9B	67.6	0.0 (−67.6)	63.3 (−4.3)	6.0 (−61.7)	51.6 (−16.0)
Llama-3.1-8B	42.3	6.3 (−36.0)	30.7 (−11.6)	56.9 (+14.6)	54.2 (+11.9)
GPT-OSS-20B	57.8	0.9 (−56.9)	30.0 (−27.7)	60.9 (+3.2)	52.2 (−5.5)
Ministral-3-8B	74.2	35.9 (−38.3)	76.9 (+2.7)	31.4 (−42.8)	68.7 (−5.5)
Full problem statement					
Model	Baseline	Explicit	Repetition	Syntax policy	Verification
Qwen3.5-9B	72.8	0.1 (−72.7)	72.4 (−0.3)	11.1 (−61.7)	54.3 (−18.5)
Llama-3.1-8B	82.2	24.0 (−58.2)	49.0 (−33.3)	79.1 (−3.2)	83.5 (+1.3)
GPT-OSS-20B	57.0	4.6 (−52.4)	31.1 (−25.9)	92.8 (+35.8)	52.5 (−4.5)
Ministral-3-8B	85.4	62.4 (−23.0)	82.2 (−3.2)	48.0 (−37.4)	71.4 (−14.0)

Table 17: Per-model candidate-normalized q (%) for RQ2. Parentheses give the paired percentage-point change from the same model–context baseline; the main table reports pair-bootstrap intervals for the four-model macro estimates.

E RQ4 Representational Analysis and Complete Summaries

E.1 Probability decomposition

For every collision item, prompt variant, and model, we retain the complete-sequence log probability of the literal surface and the log-sum probability of the two prespecified attractor surfaces. Each surface includes the model's end-of-turn token. We first average prompt variants within item–context cells, take paired differences within item, and only then average the fixed item set and four-model panel. Thus, the decomposition obeys

$$\Delta M = \Delta \log p(A) - \Delta \log p(L) \tag{5}$$

exactly at the reported aggregation level. RQ1 comparisons use neutral copy as the paired reference. RQ2 comparisons use the frozen baseline for the same context and item. The RQ2 macro row in Table 4 gives short and full contexts equal weight.

E.2 Native-template tokenization audit

We locate the literal span inside the exact tokenized chat-template prompt by verified decoded-subsequence alignment. We record the span token IDs, span length, complete literal-answer token IDs, and both complete attractor-answer tokenizations. The audit covers all three neutral-copy, five short-specification, and three full-statement RQ1 variants: 154 prompts per tokenizer. No prompt is retokenized with a model-independent tokenizer.

Model	Invariant items	Matched pairs	Pair–prompt matches
Qwen3.5-9B	14/14	7/7	77/77
Llama-3.1-8B	14/14	7/7	77/77
GPT-OSS-20B	14/14	7/7	77/77
Ministral-3-8B	14/14	7/7	77/77

Table 18: Native-template tokenization checks. "Invariant" means that an item's literal span has the same token IDs in every audited prompt. A pair–prompt match additionally requires equal literal-span length and equal literal and attractor answer lengths for the collision and its control.

Model	Context	Collision shift	Collision–control DID
Qwen3.5-9B	Short	18.39 [15.86, 19.83]	15.61 [12.65, 17.47]
	Full	18.35 [16.29, 19.70]	13.20 [10.82, 14.87]
Llama-3.1-8B	Short	25.09 [20.69, 28.68]	15.13 [11.75, 18.89]
	Full	30.25 [27.27, 32.30]	10.70 [7.19, 13.95]
GPT-OSS-20B	Short	22.55 [18.99, 25.89]	13.17 [8.18, 18.18]
	Full	21.65 [18.54, 24.95]	11.15 [6.55, 16.20]
Ministral-3-8B	Short	20.37 [18.28, 22.38]	18.26 [15.75, 20.51]
	Full	21.18 [19.07, 23.26]	14.76 [12.10, 17.41]

Table 19: RQ1 margin effects after restricting to invariant literal-span signatures. Intervals are 10,000-replicate item-bootstrap 95% intervals.

E.3 Activation-patching protocol

We run activation patching on the two BF16 checkpoints, Qwen3.5-9B and Llama-3.1-8B-Instruct. The fixed set contains seven 10^x collision items and their seven matched controls, the short- and full-specification contexts, and one prompt variant selected before inspecting patch effects. Both models have 32 decoder layers, and every layer is evaluated.

Let $h_x^{(\ell)}$ be the residual output of decoder layer ℓ at the final prompt token, which is the assistant-answer-start position under the native chat template. A patch replaces the recipient value $h_r^{(\ell)}$ with the same-position donor value $h_d^{(\ell)}$, then scores the literal and two attractor complete sequences. The primary outcome is the change in attractor-minus-literal margin. To control numerical replay error, every donor patch is corrected by subtracting the change from an identity patch at the same item and layer. Mean absolute identity changes range from 0.08 to 0.20 log-probability units across model–context cells; the corresponding 95th percentiles range from 0.20 to 0.51.

The seven frozen directions are neutral-into-baseline, explicit-disambiguation-into-baseline, self-verification-into-baseline, baseline-into-disambiguation (direction-reversal check), baseline identity, disambiguation identity, and shuffled-item explicit-disambiguation-into-baseline. This produces

$$2 \text{ models} \times 14 \text{ items} \times 2 \text{ contexts} \times 32 \text{ layers} \times 7 \text{ directions} = 12{,}544$$

patch rows. We aggregate layers into four equal relative-depth bins fixed before analysis and bootstrap items 10,000 times. Peak layers are retained only as descriptive artifacts.

Model	Context	Neutral → base	Direct → base	Verify → base	Base → direct
Qwen3.5-9B	Short	-0.49 [$-0.55, -0.43$]	-0.24 [$-0.31, -0.18$]	$+0.04$ [$-0.08, +0.15$]	$+0.67$ [$+0.55, +0.81$]
	Full	-0.78 [$-0.89, -0.68$]	-0.25 [$-0.40, -0.12$]	-0.19 [$-0.36, -0.06$]	$+0.69$ [$+0.54, +0.86$]
Llama-3.1-8B	Short	-12.04 [$-14.90, -8.66$]	-3.30 [$-4.28, -2.24$]	$+1.23$ [$+0.89, +1.58$]	$+3.21$ [$+2.41, +4.03$]
	Full	-17.77 [$-19.76, -15.75$]	-2.89 [$-4.05, -1.52$]	-0.37 [$-0.97, +0.26$]	$+2.77$ [$+1.53, +3.87$]

Table 20: Identity-corrected collision effects in the prespecified 75–100% depth bin. Entries are changes in attractor-minus-literal complete-sequence margin with item-bootstrap 95% intervals. Negative favors the literal.

For Llama, the explicit-disambiguation donor's late-bin effect is more negative than a shuffled collision donor by -7.74 [$-11.99, -5.21$] in short specifications and -3.18 [$-4.41, -2.20$] in full statements. Qwen's same comparisons are -0.02 [$-0.04, +0.01$] and -0.04 [$-0.10, +0.02$]. This contrast suggests item-specific late information at the patched Llama position but not at the corresponding Qwen position. However, explicit disambiguation also moves some matched controls, and collision-minus-control patch DID is mixed across layers. We therefore interpret the patch as a causal readout of a combined item and instruction state, not as a selective override circuit.

F RQ3 Frontier-Model Feasibility-Separation Problems

This appendix gives the ten complete English problem statements as shown in the collision $\leq$ `105` arm. RQ3 is a controlled feasibility-separation evaluation on frontier code models rather than an

estimate of average contest-task degradation. The executable experiment uses four surface variants, summarized below. The experimental manifest applies the corresponding surface to the designated collision bound(s) while otherwise preserving the problem text, formatting, and examples. Each selected task has an exact literal-scale reference solution feasible near $n \approx 10^2$, while no known exact algorithm meeting standard competitive-programming resource limits is available for the selected formulation at $n \approx 10^5$. This property is part of the diagnostic construction: it makes entry into the large-scale regime observable through abandonment of the valid literal-scale solution family. The explicit-10^5 arm is therefore used as an attractor-regime behavioral reference rather than as a large-n correctness baseline.

Arm	Surface	Role
Literal-scale reference	≤ 100	small-n feasibility reference
Matched decimal control	≤ 145	same-length non-collision control
Collision	≤ 105	target fused surface
Explicit attractor	$\leq 10^5$	hypothesized reconstructed reading

Table 21: The four surface variants used in RQ3.

F.1 Progressive Calibration

A laboratory has n calibrators. Calibrator i has reading x_i. You must choose an order in which to activate all calibrators.

After the first k calibrators have been activated, the instability of the system is the difference between the maximum and minimum readings among those k calibrators. The instability at stage k is multiplied by the given nonnegative weight w_k.

If the activation order is $p_1, p_2, \ldots, p_n$, its total cost is

$$\sum_{k=1}^{n} w_k \left(\max_{1 \leq j \leq k} x_{p_j} - \min_{1 \leq j \leq k} x_{p_j} \right).$$

Find the minimum total cost over all activation orders.

Input. The first line contains one integer n. The second line contains n integers $x_1, x_2, \ldots, x_n$. The third line contains n integers $w_1, w_2, \ldots, w_n$.

Output. Print one integer: the minimum possible total cost.

Constraints. $2 \leq n \leq 105$; $0 \leq x_i \leq 10^9$; $0 \leq w_i \leq 10^6$. All input values are integers.

Example.

```
Input
4
1 4 6 10
1 2 3 4
Output
55
```

F.2 Dependent Deployments

A laboratory can deploy any subset of n modules. Module i normally costs c_i units to deploy. When module i is deployed, its rebate may instead be activated, reducing its cost by d_i.

The rebate dependencies form a rooted tree with module 1 as the root. For every module $i > 1$, the rebate of module i may be activated only if the rebate of its parent p_i is also activated. A module whose rebate is not activated may still be deployed at its normal cost, regardless of which other modules are deployed.

Given a budget B, find the maximum number of modules that can be deployed without exceeding the budget.

Input. The first line contains two integers n and B. The second line contains two integers c_1 and d_1. Each of the next $n-1$ lines contains three integers c_i, d_i, and p_i, for i from 2 to n. The parent satisfies $1 \le p_i < i$.

Output. Print one integer: the maximum possible number of deployed modules.

Constraints. $1 \le n \le 105$; $0 \le B \le 10^9$; $1 \le c_i \le 10^6$; $0 \le d_i \le c_i$. The parent relations form a tree rooted at module 1. All input values are integers.

Example.

```
Input
5 13
8 3
6 4 1
7 1 1
5 3 2
4 0 2
Output
4
```

F.3 Three-Terminal Activation

A communication network consists of n stations joined by $n-1$ bidirectional links. The network is a tree.

Choose three distinct stations. Activating them also activates every station and link in the smallest connected subgraph containing all three chosen stations. Two choices are different if their sets of three terminal stations are different.

Given an integer d, count how many choices activate exactly d stations.

Input. The first line contains two integers n and d. Each of the next $n-1$ lines contains two integers u and v, denoting a bidirectional link between stations u and v.

Output. Print one integer: the number of three-terminal choices whose activated subgraph contains exactly d stations.

Constraints. $3 \le d \le n \le 105$; $1 \le u, v \le n$. The given links form a tree. All input values are integers.

Example.

```
Input
6 4
1 2
2 3
2 4
4 5
4 6
Output
10
```

F.4 Common Observation Trace

Two experiments produce integer sequences $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$.

A trace X must be a subsequence of both A and B. In addition, every value in X after the first one must exceed the preceding value by at least Δ. If X has length k, then $X_{i+1} - X_i \geq \Delta$ for every $1 \leq i < k$. Find the maximum possible length of such a trace.

Input. The first line contains three integers n, m, and Δ. The second line contains n integers $A_1, A_2, \ldots, A_n$. The third line contains m integers $B_1, B_2, \ldots, B_m$.

Output. Print one integer: the maximum possible length of a valid common trace.

Constraints. $1 \leq n, m \leq 105$; $1 \leq \Delta \leq 10^9$; $0 \leq A_i, B_i \leq 10^9$. All input values are integers.

Example.

```
Input
5 6 3
1 4 2 7 10
2 1 4 8 7 10
Output
4
```

F.5 Batch Variation Plans

There are n distinct samples. Sample i has measurement a_i. A processing plan partitions all samples into one or more nonempty, unlabeled batches.

The variation of a batch is the maximum measurement in that batch minus the minimum measurement in that batch. The total variation of a plan is the sum of the variations of all its batches.

Given an integer K, count the processing plans whose total variation is at most K. Two plans are different if some pair of samples belongs to the same batch in one plan and to different batches in the other. Print the answer modulo $10^9 + 7$.

Input. The first line contains two integers n and K. The second line contains n integers $a_1, a_2, \ldots, a_n$.

Output. Print one integer: the number of valid processing plans modulo $10^9 + 7$.

Constraints. $1 \leq n \leq 105$; $0 \leq K \leq 100$; $0 \leq a_i \leq 10^9$. All input values are integers.

Example.

```
Input
4 3
1 3 4 8
Output
5
```

F.6 Symbol Erasure

You are given a string s of length n consisting of lowercase English letters. Every letter has a positive erasure cost.

In one operation, you may choose a nonempty contiguous substring of the current string, provided that all characters in the chosen substring are equal. Delete the whole substring and pay the erasure cost of its character. After the deletion, the remaining parts on the left and right are concatenated, so equal characters that were previously separated may become adjacent. Find the minimum total cost required to erase the entire string.

Input. The first line contains one integer n. The second line contains the string s. The third line contains 26 integers $\mathrm{cost}[a], \mathrm{cost}[b], \ldots, \mathrm{cost}[z]$ in alphabetical order.

Output. Print one integer: the minimum total erasure cost.

Constraints. $1 \le n \le 105$; $|s| = n$; s consists only of lowercase English letters; $1 \le \mathrm{cost}[x] \le 10^6$.

Example.

```
Input
5
abaca
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Output
6
```

F.7 Three-Type Component Assembly

A warehouse contains red, green, and blue components. Their qualities are listed in arrays R, G, and B.

One assembly operation selects two unused components of different types. If their qualities are x and y, the operation earns xy points. Both selected components are then consumed and cannot be used again. You may perform any number of operations and may leave some components unused. Find the maximum total score.

Input. The first line contains three integers r, g, and b: the numbers of red, green, and blue components. The second line contains r integers $R_1, R_2, \ldots, R_r$. The third line contains g integers $G_1, G_2, \ldots, G_g$. The fourth line contains b integers $B_1, B_2, \ldots, B_b$.

Output. Print one integer: the maximum possible total score.

Constraints. $1 \le r, g, b \le 105$; $1 \le R_i, G_i, B_i \le 10^6$. All input values are integers.

Example.

```
Input
2 2 1
4 2
3 1
5
Output
26
```

F.8 Main Crate and Insulated Compartment

There are n cargo items. Item i has weight a_i.

First, select some items for a main crate so that their total weight is exactly K. Then select any subset of the items in the main crate for its insulated compartment. Every item in the insulated compartment must also belong to the main crate.

For every integer x with $0 \le x \le K$, determine whether there is a selection for which the main crate has total weight K and the insulated compartment has total weight x. Output all possible values of x in increasing order.

Input. The first line contains two integers n and K. The second line contains n integers $a_1, a_2, \ldots, a_n$.

Output. The first line contains one integer q: the number of possible values of x. If $q > 0$, the second line contains the q possible values in increasing order.

Constraints. $1 \le n, K \le 105$; $1 \le a_i \le K$. All input values are integers.

Example.

```
Input
4 7
2 3 4 5
Output
6
0 2 3 4 5 7
```

F.9 Neighbor Calibration

There are n devices arranged in a line. Device i currently has parameter a_i.

In one operation, choose two adjacent devices and replace the parameter of either one with the current parameter of the other. For some $1 \le i < n$, you may perform either $a_i = a_{i+1}$ or $a_{i+1} = a_i$. The result of an operation affects all later operations.

You may perform at most K operations. Find the minimum possible sum of all device parameters afterward.

Input. The first line contains two integers n and K. The second line contains n integers $a_1, a_2, \ldots, a_n$.

Output. Print one integer: the minimum possible sum after at most K operations.

Constraints. $1 \le n \le 105$; $0 \le K \le \min(n-1, 105)$; $0 \le a_i \le 10^9$. All input values are integers.

Example.

```
Input
5 2
5 1 4 2 3
Output
8
```

F.10 Cyclic Competition System

A culture initially contains R robots of type R, S robots of type S, and P robots of type P.

Their competition rules are cyclic: when type R meets type S, one type-S robot disappears; when type S meets type P, one type-P robot disappears; and when type P meets type R, one type-R robot disappears.

The base rates of these three effective encounters are α, β, and γ. If the current population is (r, s, p), the relative weights of the three possible effective events are αrs, βsp, and γpr, respectively. The next event is selected randomly in proportion to these weights. Encounters between robots of the same type do not change the system and are ignored.

The process continues until only one type remains. Find the probabilities that the surviving type is R, S, and P, respectively.

Input. The first line contains three integers R, S, and P. The second line contains three integers α, β, and γ.

Output. Print three real numbers: the probabilities that the surviving type is R, S, and P, in that order. An answer is accepted if the absolute or relative error of every value does not exceed 10^{-9}.

Constraints. $1 \le R, S, P \le 105$; $1 \le \alpha, \beta, \gamma \le 1000$. All input values are integers.

Example.

```
Input
1 1 1
1 2 3
Output
0.333333333333 0.500000000000 0.166666666667
```

G RQ3 Illustrative Generated Programs

This appendix presents manually selected, readily interpretable paired examples for four representative models in Table 3. The examples are explanatory case studies rather than an additional estimator: the aggregate results continue to use all ten problems and all five samples per arm. Each pair uses the same problem and sample index, and the reported score is the number of common-support tests passed out of 100. Listings reproduce the relevant generated lines verbatim; comments containing ellipses mark omitted input, initialization, or output code.

G.1 DeepSeek-V4-Pro: exact enumeration replaced by an unfinished large-n shortcut

For *Three-Terminal Activation*, sample 2 passes 100/100 tests under $n \le 100$ and 41/100 under $n \le 105$. In the control arm, the model computes all-pairs tree distances and enumerates all triples. For three terminals in a tree, half the sum of their pairwise distances is the number of edges in their minimal connecting subtree:

```
const int MAXN = 100;
int dist[MAXN][MAXN];

for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (dist[i][k] + dist[k][j] < dist[i][j])
                dist[i][j] = dist[i][k] + dist[k][j];

for (int a = 0; a < n; ++a)
    for (int b = a + 1; b < n; ++b)
        for (int c = b + 1; c < n; ++c) {
            int edges = (dist[a][b] + dist[b][c]
                       + dist[a][c]) / 2;
            if (edges + 1 == d) ++count;
        }
```

With only the displayed bound changed to 105, the generated program instead sizes its tree and binary-lifting tables for 100,000 vertices:

```
const int MAXN = 100000;
const int LOG = 17; // sufficient for the maximum input size
int head[MAXN + 1], to[2 * MAXN], nxt[2 * MAXN];
int depth[MAXN + 1], parent[LOG][MAXN + 1];

for (int u = 1; u <= n; u++) {
    // ... collect branches incident to u ...
    long long cnt[2 * MAXN] = {};
    for (int i = 0; i < cntBranches; i++) {
        // This is simplified here -- in full solution you'd DFS
        // from u to count nodes at each distance.
    }
}
printf("%lld\n", ans);
```

The large-n branch-counting idea is never implemented and ans is never updated, so the program always prints zero. It passes only instances whose correct answer happens to be zero. This pair directly exposes a feasibility-driven change in algorithm design without relying on a retired benchmark problem.

G.2 GLM-5.2: INTERVAL STATE REPLACED BY AN INVALID COMPRESSED STATE

For *Symbol Erasure*, sample 4 changes from 100/100 to 4/100. Under $n \leq 100$, GLM generates a memoized interval recurrence that explicitly represents deletion of the interior between two equal symbols:

```
long long dp[105][105];
bool computed[105][105];

long long solve(int l, int r) {
    if (l > r) return 0;
    if (l == r) return cost[s[l] - 'a'];
    if (computed[l][r]) return dp[l][r];
    computed[l][r] = true;

    long long best = cost[s[l] - 'a'] + solve(l + 1, r);
    for (int k = l + 1; k <= r; k++) {
        if (s[k] == s[l]) {
            long long cur = solve(l + 1, k - 1) + solve(k, r);
            if (cur < best) best = cur;
        }
    }
    dp[l][r] = best;
    return best;
}
```

Under $n \leq 105$, the generated arrays are sized for 100,000 positions and the two-dimensional interval state is replaced by a position–character summary:

```
const int MAXN = 100005;
const int ALPHABET = 26;
int best[MAXN][ALPHABET];
int prev_dp[MAXN];
int curr_dp[MAXN];

for (int j = 1; j < n; ++j) {
    for (int i = 0; i + j < n; ++i) {
        int c = s[i + j] - 'a';
        int res = prev_dp[i] + cost[c];
        int match_cost = best[i][c] + prev_dp[i + 1];
        if (match_cost < res) res = match_cost;
        curr_dp[i] = res;
    }
    // ... update best and copy curr_dp to prev_dp ...
}
```

This compressed state does not retain the two interval boundaries required to account for arbitrary sequences of interior deletions, explaining the near-total executable failure.

G.3 GROK-4.6: THREE-DIMENSIONAL DP REPLACED BY LOCAL PAIRING

For *Three-Type Component Assembly*, sample 3 changes from 100/100 to 14/100. The ≤ 100 program uses the intended three-dimensional DP and considers all three possible pair types:

```
int R[105], G[105], B[105];
long long dp[105][105][105];

for (int i = 0; i <= r; i++) {
    for (int j = 0; j <= g; j++) {
        for (int k = 0; k <= b; k++) {
            long long val = 0;
            if (i > 0) val = mx(val, dp[i - 1][j][k]);
            if (j > 0) val = mx(val, dp[i][j - 1][k]);
            if (k > 0) val = mx(val, dp[i][j][k - 1]);
            if (i > 0 && j > 0) val = mx(val, dp[i - 1][j - 1][k] + 1LL * R[i] * G[j]);
            if (i > 0 && k > 0) val = mx(val, dp[i - 1][j][k - 1] + 1LL * R[i] * B[k]);
            if (j > 0 && k > 0) val = mx(val, dp[i][j - 1][k - 1] + 1LL * G[j] * B[k]);
            dp[i][j][k] = val;
        }
    }
}
```

The ≤ 105 program instead allocates arrays for 100,000 elements and repeatedly pairs the two largest currently available values of different types:

```
const int MAXN = 100005;
int R[MAXN], G[MAXN], B[MAXN];

while (true) {
    int vr = (i < r ? R[i] : -1);
    int vg = (j < g ? G[j] : -1);
    int vb = (k < b ? B[k] : -1);

    if (vr >= vg && vr >= vb) {
        if (vg >= vb && vg >= 0) {
            ans += (long long)vr * vg; i++; j++;
        } else if (vb >= 0) {
            ans += (long long)vr * vb; i++; k++;
        } else break;
    }
    // ... symmetric cases for green and blue ...
}
```

The locally largest product need not be globally optimal because consuming two types can prevent two later high-value cross-type pairs; the full DP state is precisely what captures this trade-off.

G.4 SEED-2.0-CODE: INTERVAL DP REPLACED BY COUNTING VISIBLE BLOCKS

For *Symbol Erasure*, sample 4 changes from 100/100 to 19/100. The ≤ 100 program constructs an interval DP and minimizes over split points:

```
long long dp[105][105];

for (int l = 2; l <= n; ++l) {
    for (int i = 0; i + l <= n; ++i) {
        int j = i + l - 1;
        dp[i][j] = INF;
        if (s[i] == s[j])
            dp[i][j] = min(dp[i+1][j], dp[i][j-1]);
        for (int k = i; k < j; ++k) {
            long long cur = dp[i][k] + dp[k+1][j];
            if (s[k] == s[j])
                cur -= min(cost[s[k]-'a'], cost[s[j]-'a']);
            if (cur < dp[i][j]) dp[i][j] = cur;
        }
    }
}
```

The ≤ 105 response explicitly says that n may be 100,000, claims an $O(n)$ solution, and pays once for every block visible in the original string:

```
long long total = 0;
char current = s[0];
for (int i = 1; i < n; ++i) {
    if (s[i] != current) {
        total += cost[current - 'a'];
        current = s[i];
    }
}
total += cost[current - 'a'];
```

This omits the defining operation of the problem: erasing an interior block can concatenate two equal outer blocks and allow them to be erased together. For example, with string `aba`, deleting `b` first can merge the two `a` symbols, whereas counting the initially visible blocks always charges for two separate `a` blocks.

H RQ3 UNCERTAINTY AND PER-PROBLEM RESULTS

We bootstrap at the problem level. For each model, 10,000 paired resamples draw ten problem identifiers with replacement and retain the paired ≤ 100 and ≤ 105 conditions and all five generations for every selected problem. Table 22 reports the resulting intervals. Completed answers without extractable code and requests that remain incomplete after retries receive zero executable credit in every resample.

Model	Metric	≤ 100 [95% CI]	≤ 105 [95% CI]	Δ [95% CI]
		Reasoning Models		
GPT-5.6-Sol (medium)	Avg@5	100 [100,100]	48 [34,62]	-52 [$-66, -38$]
	Test-case Avg@5	100 [100,100]	55.06 [41.24,67.92]	-44.94 [$-58.76, -32.08$]
Grok-4.6	Avg@5	100 [100,100]	92 [84,100]	-8 [$-16, 0$]
	Test-case Avg@5	100 [100,100]	95.80 [91.32,100]	-4.20 [$-8.68, 0$]
Gemini-3.7-Flash	Avg@5	98 [94,100]	70 [46,90]	-28 [$-54, -6$]
	Test-case Avg@5	98.94 [96.82,100]	84.28 [70.72,94.88]	-14.66 [$-28.78, -3.04$]
		Non-reasoning Models		
DeepSeek-V4-Pro	Avg@5	66 [42,88]	12 [4,22]	-54 [$-72, -36$]
	Test-case Avg@5	76.72 [60.12,91.90]	28.20 [19.06,37.56]	-48.52 [$-62.72, -31.82$]
GLM-5.2	Avg@5	60 [46,74]	14 [4,24]	-46 [$-60, -30$]
	Test-case Avg@5	71.40 [59.08,83.34]	37.66 [26.94,48.06]	-33.74 [$-44.66, -21.42$]
Seed-2.0-Code	Avg@5	58 [40,74]	2 [0,6]	-56 [$-74, -38$]
	Test-case Avg@5	71.78 [57.88,83.58]	24.88 [13.16,37.76]	-46.90 [$-64.86, -28.84$]

Table 22: RQ3 problem-cluster bootstrap intervals. All entries are percentages; Δ is the paired `105`-minus-100 change.

Table 23 gives the paired Avg@5 change for each problem. Negative values indicate fewer fully correct generations under the `105` surface. The pattern spans multiple tasks and models, showing that the aggregate collision effect is distributed across the benchmark rather than driven by a single problem.

	Reasoning Models			Non-reasoning Models		
Problem	GPT	Grok	Gemini	DeepSeek	GLM	Seed
Common Observation Trace	-20	0	-20	-100	-80	-20
Cyclic Competition	-40	0	0	-80	-20	-40
Dependent Deployments	-80	0	$+20$	-20	0	-20
Batch Variation Plans	-80	-20	-100	0	-20	-20
Neighbor Calibration	-80	0	-100	-40	-60	-80
Nested Cargo Selection	-20	0	0	-60	-60	-60
Progressive Calibration	-80	0	-20	-60	-40	-80
Symbol Erasure	-40	-40	-40	-40	-60	-80
Three-Type Component Assembly	-40	-20	0	-40	-60	-60
Three-Terminal Activation	-40	0	-20	-100	-60	-100

Table 23: Per-problem paired change in Avg@5 (percentage points). Negative values indicate lower performance under the $\leq$ `105` condition.

H.1 Systematic Answer Audit

We use DeepSeek-V4-Flash as a fixed answer auditor. Given the problem, the intended asymptotic complexity, and one generated response, it classifies the implemented algorithm as the intended reference family, a lower-complexity shortcut, another algorithm, or unclear. It separately records whether the explanation and implementation make mutually consistent complexity assumptions. Table 24 reports this auxiliary audit for all six model panels, while executable correctness is measured independently by compilation and testing.

The audit reveals the clearest shift in the non-reasoning panels: under the `105` surface, reference-family solutions become substantially less frequent while explanation–implementation inconsistencies become more frequent. The reasoning panels change less, matching their stronger executable performance in Table 3.

	Intended reference family (%)			Explanation–implementation inconsistent (%)		
Model	≤ 100	≤ 105	Δ	≤ 100	≤ 105	Δ
Reasoning Models						
GPT-5.6-Sol	96.0	70.0	-26.0	0.0	10.0	$+10.0$
Grok-4.6	90.0	88.0	-2.0	2.0	6.0	$+4.0$
Gemini-3.7-Flash	92.0	82.0	-10.0	4.0	12.0	$+8.0$
Non-reasoning Models						
DeepSeek-V4-Pro	88.0	48.0	-40.0	8.0	64.0	$+56.0$
GLM-5.2	90.0	40.0	-50.0	6.0	56.0	$+50.0$
Seed-2.0-Code	89.8‡	10.0	-79.8	14.3‡	76.0	$+61.7$

Table 24: Auxiliary systematic audit of generated RQ3 answers. Each arm contains 50 samples. ‡One Seed control-arm answer lacks an auditable final response, so its ≤ 100 denominator is 49.