SAGE: SELF-ATTRIBUTION-GUIDED EVOLVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt evolution optimizes the instruction given to a frozen language model: it maintains a population of candidate instructions and repeatedly mutates, evaluates, and selects them. Existing prompt optimizers select candidates by task accuracy alone, but accuracy cannot tell whether a correct answer is due to the instruction. Two instructions can be equally accurate even when one induces a distinct solution strategy and the other merely rewords an existing candidate and leads the model to behave the same way. We formalize this blind spot: among candidates whose rewards follow the same distribution, the rewards carry no information about which candidate produced them, whereas the model's trajectories do. To recover this information, we introduce *self-attribution*, which measures how much better a candidate's own instruction explains its trajectory than the best competing instruction does, and we prove that it yields a computable lower bound on the information that accuracy discards. We then propose **SAGE** (Self-Attribution-Guided Evolving), which adds this signal to two decisions of GEPA, a state-of-the-art prompt optimizer: which candidates are selected as parents, and which mutated children are kept. Across four open-weight models spanning a $30\times$ range in size and six tasks, SAGE achieves the highest average accuracy on every model and improves over GEPA by up to 4.0 points. It also ends below its starting prompt in only 3 of 24 model–task pairs, compared with 12 for GEPA. These gains persist on harder out-of-distribution problems, under cross-model prompt transfer, and with a black-box API model whose self-attribution is computed by an open-weight surrogate.

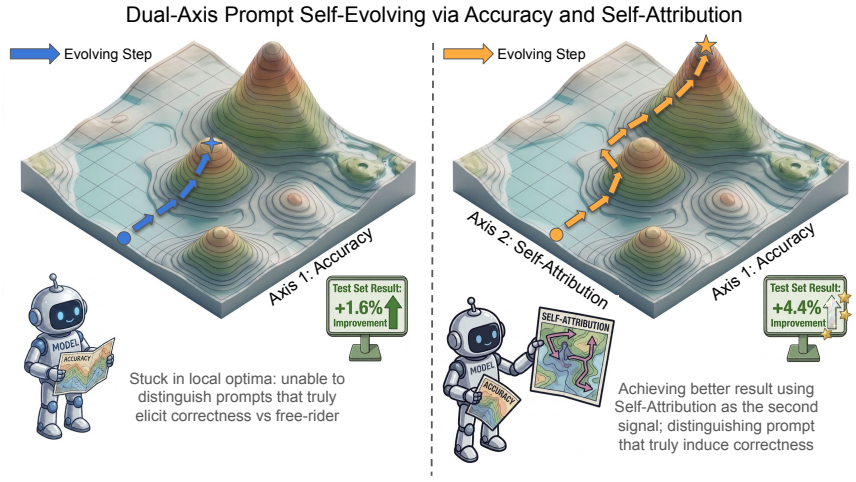


Figure 1: **Prompt evolution with one signal versus two.** Guided by accuracy alone (left), an optimizer cannot separate equally accurate instructions and stalls in a local optimum. SAGE (right) also uses self-attribution: whether an instruction explains its own trajectory better than every other candidate does, which accuracy provably cannot reveal (§2.2).

1 INTRODUCTION

Large language models can be steered through their prompts alone (Brown et al., 2020; Wei et al., 2022; Schulhoff et al., 2025), and *prompt evolution* automates the search for a good prompt. Given

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.

a frozen model, a prompt optimizer maintains a population of candidate instructions and repeatedly mutates, evaluates, and selects them, keeping the instructions that perform well and deriving new candidates from them (Zhou et al., 2023; Pryzant et al., 2023; Yang et al., 2024; Fernando et al., 2023; Guo et al., 2024; Wang et al., 2024a; Opsahl-Ong et al., 2024; Agrawal et al., 2026). These optimizers select candidates by a single signal: task accuracy, that is, whether the model's output is correct (Opsahl-Ong et al., 2024; Yuksekgonul et al., 2024; Agrawal et al., 2026; Zhang et al., 2026b). Accuracy is the natural fitness signal, and on hard problems it is an informative one. Because it is the only signal, however, the search cannot see a property that matters most once the search starts to succeed.

The missing property is whether a candidate's correctness is due to its own instruction. Two instructions can be correct on the same problems and still differ in value. The first induces its own behavior: the model solves the problems in a way that follows from this instruction's wording. The second is a reworded copy, or *free rider*: the model would produce the same trajectories under other candidates in the population (Webson & Pavlick, 2022; Min et al., 2022). The main job of the search is to decide which candidates to keep, select as parents, and mutate, yet accuracy scores these two candidates identically. The problem grows as the search progresses, because a strong optimizer fills its population with candidates that all score well, leaving accuracy with less and less to distinguish.

We propose *self-attribution* as a second signal for prompt evolution, complementary to accuracy. For each candidate, it measures how much better the candidate's own instruction explains its correct trajectory than any competing instruction does; in other words, it asks whether the behavior can be attributed to this instruction. A candidate with high self-attribution produces trajectories that point to its own instruction, so its correctness is evidence that the instruction matters. A candidate with low self-attribution is a copy whose success another candidate explains equally well. The signal is informative throughout the search, not only after accuracy saturates, although it matters most there. In §2, we prove that among candidates whose rewards follow the same distribution, accuracy carries zero information about which candidate is which while the trajectories carry a positive amount, and that self-attribution gives a computable lower bound on this gap.

To test whether self-attribution helps in practice, we build **SAGE** (Self-Attribution-Guided Evolving) on top of GEPA (Agrawal et al., 2026), a state-of-the-art prompt optimizer. SAGE uses the signal in two of GEPA's decisions: which candidates are selected as parents, and which mutated children are kept. We evaluate on four open-weight models spanning a $30\times$ range in size and on six tasks covering competition mathematics, multi-domain reasoning, symbolic planning, and text-to-SQL. All optimizers receive the same budget of metric calls, and computing the self-attribution signal adds only about 5% to wall-clock time (Appendix C). SAGE achieves the highest average accuracy on every model and improves over GEPA by up to 4.0 points. It is also more reliable: GEPA ends below its starting prompt on 12 of the 24 model–task pairs, and SAGE on only 3. The advantage persists on harder problems, under cross-model transfer, and with a black-box API model whose self-attribution is computed by an open-weight surrogate.

Our contributions are as follows. *(i)* We identify a blind spot of accuracy-only prompt evolution and propose self-attribution as a second, complementary selection signal (§2). *(ii)* We prove that accuracy cannot distinguish candidates whose rewards follow the same distribution, and that the self-attribution margin lower-bounds the information that accuracy discards (§2.4, Appendix J). *(iii)* We propose **SAGE**, which adds the self-attribution signal to a state-of-the-art prompt optimizer at two decision points (§3). *(iv)* We show that SAGE achieves the highest average accuracy on every model across six tasks, and that its advantage holds under distribution shift, cross-model transfer, and black-box deployment (§4).

2 Self-Attribution: A Second Signal for Prompt Evolution

This section defines self-attribution independently of any particular optimizer. We specify no mutation operator, acceptance rule, or archive, so the results apply to any procedure that maintains a population of candidate instructions for a frozen model. They also hold for any score that is a deterministic function of the model's output, so we use the general term *reward* in this section and specialize to accuracy in the experiments.

2.1 SETUP

Fix a *frozen* model π, a problem x, and a population of candidate instructions $\Phi_1, \ldots, \Phi_n$. Given the problem, instruction Φ_j induces a distribution $P_j := \pi(\cdot \mid \mathrm{chat}(x, \Phi_j))$ over trajectories (the model's full responses), from which we draw one rollout $Z_j \sim P_j$. Because π is fixed, every quantity below is a deterministic function of a (context, string) pair and can be cached. A reward function that is deterministic in the trajectory, such as a verifier, a rubric, a count of passing tests, or answer accuracy (Zheng et al., 2023; Lambert et al., 2025; Guo et al., 2025), scores each rollout, and collecting these scores gives the reward matrix ρ. A reward-driven optimizer sees only ρ: it selects parents, accepts children, and returns its final prompt using functions of ρ alone.

2.2 WHAT REWARD CANNOT SEE

Reward summarizes behavior, and the summary loses information. We call a set $\mathcal{E}$ of $N \geq 2$ candidates an *equal-reward-law class* when their rewards follow the same distribution: for example, every candidate always answers correctly, every candidate always answers incorrectly, or all candidates answer correctly with the same probability. Such classes arise naturally as the search succeeds, because a strong optimizer pushes its population toward candidates that all score well; Remark J.10 states when they are guaranteed to exist. Within such a class, the rewards give no way to tell the candidates apart: any rule that reads only the rewards identifies a candidate no better than a random guess. A reward-only optimizer therefore cannot separate a candidate with its own strategy from a free rider.

2.3 THE SELF-ATTRIBUTION MARGIN

The information that reward discards is still present in the trajectories, and the frozen model can recover it cheaply. The *self-attribution score* of a trajectory z under instruction Φ_k is its teacher-forced log-likelihood $\ell^{\mathrm{tot}}(z; \Phi_k) := \log \pi(z \mid \mathrm{chat}(x, \Phi_k))$, which measures how well Φ_k explains the behavior z. Scoring the population's n rollouts under all n instructions fills the *cross-score matrix* $L[j, k] := \ell^{\mathrm{tot}}(z_j; \Phi_k)$. This takes n^2 prefill-only forward passes, with no decoding, no gradients, and no reward model, and the results can be cached permanently because π is frozen. From L we compute the *self-attribution margin* $m_j^{\mathrm{tot}} := L[j, j] - \max_{k \neq j} L[j, k]$: the score of candidate j's own instruction on its rollout minus the best competing instruction's score on the same rollout. We use total rather than per-token log-likelihoods throughout. Because both terms score the same trajectory, properties of the trajectory itself, such as its length or the difficulty of the problem, do not favor either instruction. A positive margin means that Φ_j explains its own behavior better than any other instruction in the population does. A free rider cannot earn one: if a candidate behaves exactly like another candidate, its margin is at most zero almost surely (Lemma J.15).

2.4 WHY SELF-ATTRIBUTION IS INFORMATIVE

Two results, proved in Appendix J, show that self-attribution complements reward. First, within an equal-reward-law class, the rewards carry no information about the candidates, while their trajectories do. Let K be a uniformly random member of the class, Z its rollout, ρ_K its rewards, and $\bar{P} := \frac{1}{N} \sum_{k \in \mathcal{E}} P_k$. Then

$$I(K; \rho_K) \;=\; 0, \quad \text{while} \quad I(K; Z) \;=\; \frac{1}{N} \sum_{j \in \mathcal{E}} D_{\mathrm{KL}}(P_j \,\|\, \bar{P}) \;>\; 0 \text{ unless all } P_j \text{ coincide.} \tag{1}$$

The first equality is exact. Because the reward distributions are identical, the rewards reveal nothing about which candidate produced them, and any rule based on rewards alone identifies the candidate with probability $1/N$ (Theorem J.9 and Corollary J.7). The second quantity, the generalized Jensen–Shannon divergence of the candidates' trajectory distributions, is positive unless all candidates behave identically. The information needed to tell the candidates apart therefore exists, but only in the trajectories; since the reward is a function of the trajectory, an optimizer that reads only rewards loses all of it. Second, the margin gives a computable lower bound on this information. Reading each row of L within the class as a guess of which instruction generated the trajectory gives, with no assumption on the sign or size of the margins,

$$I(K; Z) \;\geq\; \log N - \mathbb{E}\big[\varphi_N(m_K^{\mathrm{tot}})\big], \qquad \varphi_N(t) = \log\big(1 + (N-1)e^{-t}\big) \tag{2}$$

(Theorem J.13). The bound grows with the margins and approaches its maximum, $\log N$, as all margins become large. Margins at or below zero certify nothing; a bound of zero means that the information is not certified, not that the candidates are indistinguishable.

Self-attribution does not measure quality, however. An instruction that makes the model emit nearly the same wrong answer every time can still have the largest margin in its class (Lemma J.16), so maximizing the margin would favor the wrong candidate (Proposition J.17), a familiar failure of optimizing a proxy (Skalse et al., 2022). We therefore use self-attribution only as a threshold constraint on top of accuracy: among candidates that are already correct, we prefer those with a positive margin, and we never maximize the margin itself. SAGE is built around this rule (§3).

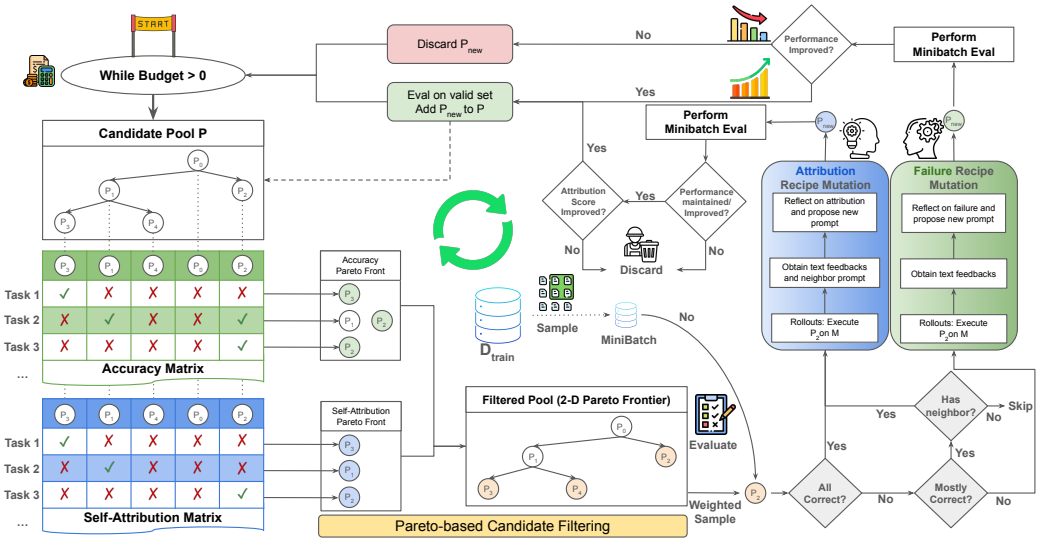


Figure 2: **Overview of SAGE.** SAGE adds the self-attribution signal to two decisions of the host optimizer (GEPA) and leaves everything else unchanged. **Parent selection (§3.1):** per-instance accuracy fronts and self-attribution fronts are merged before the host's Pareto-based filtering and weighted sampling. **Proposal and acceptance (§3.2):** a mostly correct parent with an attributional neighbor is mutated with the attribution recipe instead of the host's failure recipe; on a minibatch that the parent already solves, a child is kept only if it still solves every problem and its trajectories are attributable to its own instruction.

3 SAGE

SAGE adds the self-attribution signal to GEPA (Agrawal et al., 2026), a state-of-the-art reflective prompt optimizer that we call the *host*. SAGE keeps the host's objective and final-selection rule, and it keeps the host's accuracy-based acceptance rule except on minibatches that the parent already solves completely (§3.2). The only new information is the self-attribution margin of §2, which SAGE uses in two places. A *probe* computes the margins by running a prefill-only forward pass over each (trajectory, instruction) pair to fill the cross-score matrix L, so it costs teacher-forced scoring and no generation. Margins are computed only on correct trajectories. Two properties limit the risk of the new signal: it only ranks candidates that have already solved the problem at hand, and the returned prompt is still the candidate with the highest validation accuracy. The signal therefore changes where the search spends its budget, but not how the final prompt is chosen. Figure 2 gives an overview.

3.1 Parent selection with a self-attribution front

The host selects parents using *per-instance accuracy fronts*: for each validation problem, the set of candidates with the best score on it. It first removes *dominated* candidates, those for which every front they belong to also contains another remaining candidate, and then samples a parent

with probability proportional to the number of fronts it belongs to. SAGE adds a second family of per-instance fronts. For every problem solved by at least two candidates, the *self-attribution front* contains the candidates whose own instruction best explains their trajectory, i.e., those with $m_j^{\text{tot}} > 0$ against the other candidates that solve the problem. The two families of fronts are merged and passed to the host's pruning and sampling, which are unchanged. In practice, the change matters at the pruning step. In our runs, the host never drops a candidate because it is absent from all accuracy fronts; it drops candidates because they are dominated. A self-attribution front can give such a candidate a front on which no other remaining candidate appears, so it is no longer dominated and stays in the pool of possible parents. Appendix G.1 replays every such decision.

3.2 Proposing and accepting distinct children

Reflective mutation rewrites a prompt to fix the errors it made on a minibatch. A parent that is already mostly correct leaves little to fix, so the mutation tends to rephrase it, producing near-copies among the strongest candidates, where a different strategy would be most useful. SAGE changes the mutation in this case. It switches to a *self-attribution recipe* when the parent is mostly correct (minibatch failure rate below $\theta = 0.4$), looks like a copy (its fraction of positive margins is at or below the population median), and has an *attributional nearest neighbor*, i.e., another candidate whose instruction best explains the parent's trajectories. The reflection model sees examples that the parent already solves together with the neighbor's instruction, and rewrites the parent's instruction so that it approaches the task differently from the neighbor (Appendix K). This moves the mutation toward a strategy the population does not yet contain, instead of rephrasing one it already has.

The second case is a minibatch that the parent solves completely. Accuracy cannot improve on such a minibatch, so the host, which keeps a child only if it strictly improves minibatch accuracy, skips it and the iteration yields nothing. On this all-correct path only, SAGE replaces the host's rule with a *self-attribution acceptance gate*: a child is kept if (i) it still solves the whole minibatch, and (ii) its trajectories are better explained by its own instruction than by the parent's or the neighbor's, i.e., its mean margin over the minibatch is positive. Accuracy is checked first and is never traded away. This is the constraint form of §2.4: a positive margin is required, not maximized. An accepted child then receives the same full validation as any other candidate and joins the population on equal terms. Appendix G.2 traces these children: children of the self-attribution recipe lie on the ancestry of the returned prompt in 8 of the 20 optimization runs, 7 of them admitted through this gate.

4 Experiments

We ask three questions. Does the self-attribution signal improve on the accuracy-only optimizer, and at what risk (§4.2)? Which component of SAGE is responsible for the gain (§4.3)? Does the advantage hold in conditions the search never saw: harder problems, a different base model, and an executor that exposes no log-probabilities (§4.4)?

4.1 Setup

Models. We use four open-weight models spanning roughly $30\times$ in parameter count: Qwen3-4B-Instruct-2507 and Qwen3-30B-A3B-Instruct-2507 (Yang et al., 2025), and the reasoning models GPT-OSS-20B and GPT-OSS-120B (OpenAI et al., 2025). Each model serves as both the task executor and the reflection model, so no method benefits from a stronger external proposer. All models are served locally with vLLM (Kwon et al., 2023), from which the probe reads teacher-forced log-probabilities. Inference parameters are listed in Appendix A.2.

Datasets. We evaluate on six tasks from four families: competition mathematics with MATH500 (Hendrycks et al., 2021; Lightman et al., 2023) and the AIME 2025 and 2026 contests (Balunović et al., 2026), multi-domain multiple-choice reasoning with MMLU-Pro (Wang et al., 2024b), symbolic planning with Sokoban (Wang et al., 2025; Schrader, 2018), and text-to-SQL generation with WikiSQL (Zhong et al., 2018). Appendix A.1 gives dataset sizes, stratification, and the train/validation/test protocol.

Methods. We compare five methods. **Zero-shot CoT** is the fixed seed instruction from which every optimizer starts, evaluated without search. **Few-shot CoT** adds three in-context examples to the same seed instruction, also without search; the examples are the model's own correct reasoning traces on the training split, so no method sees demonstrations from a stronger model. **MIPROv2** (Opsahl-Ong et al., 2024) jointly optimizes the instruction and the few-shot demonstrations with Bayesian surrogate search over a bootstrapped pool of demonstrations, so its search space contains ours. **GEPA** (Agrawal et al., 2026) is the host of SAGE. It selects parents on instance-level accuracy Pareto fronts, the strongest published use of accuracy as a selection signal that we are aware of, and is therefore our main baseline. **SAGE** is GEPA with the self-attribution signal added at the two decisions of §3. Appendix A.3 gives the configuration of Few-shot CoT and MIPROv2.

Protocol. Every optimizer receives the same budget of 1800 metric calls. We report **Accuracy Average@4**: for each test problem, we draw four samples at temperature 0.7, score each with the task's metric, and average over the samples and over the problems of the held-out test split. All numbers are accuracies in percent. Remaining hyperparameters are listed in Appendix A.3.

4.2 Main results

Table 1: **Main results** (held-out test Accuracy Average@4, %). We compare SAGE with four baselines: *Zero-shot CoT*, *Few-shot CoT*, *MIPROv2*, and *GEPA*. Values in parentheses are absolute improvements (percentage points) over Zero-shot CoT on the same model; **bold** marks the best and underline the second-best method in each column within a model. SAGE has the highest average on every model and improves on GEPA on all four (SAGE−GEPA $= +3.99/+2.58/+1.85/+1.04$, models ordered by size).

		Benchmark						
Model	**Method**	**AIME'25**	**AIME'26**	**MATH500**	**MMLU-Pro**	**Sokoban**	**WikiSQL**	**Average**
Qwen3-4B	Zero-shot CoT	45.83	52.50	92.21	63.97	49.50	46.23	58.37
	Few-shot CoT	35.83 (−10.00)	44.17 (−8.33)	94.47 (+2.26)	71.13 (+7.16)	**59.25** (+9.75)	47.55 (+1.32)	58.73 (+0.36)
	MIPROv2	47.50 (+1.67)	52.50 (+0.00)	93.97 (+1.76)	**72.82** (+8.85)	51.75 (+2.25)	48.30 (+2.07)	61.14 (+2.77)
	GEPA	46.67 (+0.84)	51.67 (−0.83)	89.45 (−2.76)	71.63 (+7.66)	47.00 (−2.50)	50.13 (+3.90)	59.43 (+1.06)
	SAGE	**49.17** (+3.34)	**55.00** (+2.50)	**94.85** (+2.64)	72.13 (+8.16)	57.75 (+8.25)	**51.63** (+5.40)	**63.42** (+5.05)
GPT-OSS-20B	Zero-shot CoT	67.50	69.17	**97.49**	71.88	78.75	52.01	72.80
	Few-shot CoT	70.83 (+3.33)	**75.83** (+6.66)	97.24 (−0.25)	73.00 (+1.12)	79.62 (+0.87)	53.71 (+1.70)	75.04 (+2.24)
	MIPROv2	65.83 (−1.67)	73.33 (+4.16)	97.11 (−0.38)	70.07 (−1.81)	80.25 (+1.50)	**72.24** (+20.23)	76.47 (+3.67)
	GEPA	66.67 (−0.83)	70.00 (+0.83)	97.24 (−0.25)	71.38 (−0.50)	78.50 (−0.25)	66.52 (+14.51)	75.05 (+2.25)
	SAGE	**71.67** (+4.17)	70.00 (+0.83)	97.24 (−0.25)	**74.38** (+2.50)	**81.62** (+2.87)	70.85 (+18.84)	**77.63** (+4.83)
Qwen3-30B-A3B	Zero-shot CoT	59.17	70.83	**96.86**	79.55	58.00	52.39	69.47
	Few-shot CoT	56.67 (−2.50)	65.83 (−5.00)	96.11 (−0.75)	**81.17** (+1.62)	62.75 (+4.75)	53.14 (+0.75)	69.28 (−0.19)
	MIPROv2	**63.33** (+4.16)	**75.83** (+5.00)	96.73 (−0.13)	79.99 (+0.44)	55.38 (−2.62)	53.58 (+1.19)	70.81 (+1.34)
	GEPA	60.00 (+0.83)	71.67 (+0.84)	96.36 (−0.50)	80.42 (+0.87)	64.62 (+6.62)	52.64 (+0.25)	70.95 (+1.48)
	SAGE	**63.33** (+4.16)	75.00 (+4.17)	96.73 (−0.13)	79.93 (+0.38)	**65.88** (+7.88)	**55.90** (+3.51)	**72.80** (+3.33)
GPT-OSS-120B	Zero-shot CoT	77.50	80.00	97.74	**81.11**	91.38	51.32	79.84
	Few-shot CoT	77.50 (+0.00)	77.50 (−2.50)	97.61 (−0.13)	80.36 (−0.75)	**96.25** (+4.87)	54.90 (+3.58)	80.69 (+0.85)
	MIPROv2	78.33 (+0.83)	**83.33** (+3.33)	**98.24** (+0.50)	77.24 (−3.87)	94.00 (+2.62)	53.52 (+2.20)	80.78 (+0.94)
	GEPA	**80.83** (+3.33)	79.17 (−0.83)	97.24 (−0.50)	79.36 (−1.75)	91.00 (−0.38)	71.80 (+20.48)	83.23 (+3.39)
	SAGE	**80.83** (+3.33)	**83.33** (+3.33)	97.86 (+0.12)	79.30 (−1.81)	91.75 (+0.37)	**72.55** (+21.23)	**84.27** (+4.43)

SAGE has the highest average on every model. In the *Average* column of Table 1, SAGE ranks first on all four models. The other methods have no consistent order: MIPROv2 is second on the two smaller models but third on the two larger ones, GEPA is second only on the two larger models, and Few-shot CoT even falls below the zero-shot seed on Qwen3-30B-A3B. We consider this consistency across four independently optimized models more important than the size of any single gain, since a general-purpose optimizer should help on every model, not only on some.

SAGE improves on GEPA on every model. GEPA is the controlled comparison: SAGE is GEPA plus the self-attribution signal, with the same budget and nothing else changed. SAGE outperforms GEPA on all four models, by $+3.99$, $+2.58$, $+1.85$, and $+1.04$ points from the smallest to the largest. A paired bootstrap over test instances finds the gain significant on the three smaller models but not on GPT-OSS-120B (Appendix E). The gain shrinks as the base model grows, likely because a stronger model leaves less in-distribution headroom for any optimizer. For large models, our main

evidence is therefore the out-of-distribution and transfer results of §4.4, where SAGE's advantage does not shrink with scale.

SAGE rarely makes the starting prompt worse. An optimizer is useful only if it rarely hurts, and accuracy-only search is fragile in this respect. GEPA ends below the zero-shot seed in 12 of the 24 model–task cells, by 1.0 point on average and 11.9 points in total, and MIPROv2 does so in 6 cells. SAGE ends below the seed in only 3 cells, by 2.2 points in total. Because these counts cover all 24 cells, no single noisy cell drives them. We attribute this robustness to the form of the signal: an accuracy-gated constraint that only reorders candidates that are already good enough.

4.3 Ablation: which components help

Table 2: **Ablation of where the self-attribution signal is used** (held-out test Accuracy Average@4, %). Rows are *cumulative*: **GEPA** is the accuracy-only host; + **SA-Pareto** merges the per-instance self-attribution fronts into the host's accuracy fronts for parent selection (§3.1); + **SA-Proposer** further adds the self-attribution proposer and its acceptance gate (§3.2), which gives full SAGE. SA stands for self-attribution. Values in parentheses are improvements over GEPA; bold marks the best method in each column within a model.

		Benchmark						
Model	**Method**	**AIME'25**	**AIME'26**	**MATH500**	**MMLU-Pro**	**Sokoban**	**WikiSQL**	**Average**
Qwen3-4B	GEPA	46.67	51.67	89.45	71.63	47.00	50.13	59.43
	+ SA-Pareto	45.00 (−1.67)	50.00 (−1.67)	94.85 (+5.40)	70.57 (−1.06)	52.12 (+5.12)	47.74 (−2.39)	60.05 (+0.62)
	+ SA-Proposer (SAGE)	**49.17** (+2.50)	**55.00** (+3.33)	**94.85** (+5.40)	**72.13** (+0.50)	**57.75** (+10.75)	**51.63** (+1.50)	**63.42** (+3.99)
GPT-OSS-20B	GEPA	66.67	70.00	97.24	71.38	78.50	66.52	75.05
	+ SA-Pareto	70.83 (+4.16)	**72.50** (+2.50)	96.86 (−0.38)	72.01 (+0.63)	79.38 (+0.88)	69.28 (+2.76)	76.81 (+1.76)
	+ SA-Proposer (SAGE)	**71.67** (+5.00)	70.00	**97.24**	**74.38** (+3.00)	**81.62** (+3.12)	**70.85** (+4.33)	**77.63** (+2.58)
Qwen3-30B-A3B	GEPA	60.00	71.67	96.36	**80.42**	64.62	52.64	70.95
	+ SA-Pareto	**65.00** (+5.00)	**76.67** (+5.00)	**96.98** (+0.62)	79.68 (−0.74)	64.12 (−0.50)	**59.48** (+6.84)	**73.66** (+2.71)
	+ SA-Proposer (SAGE)	63.33 (+3.33)	75.00 (+3.33)	96.73 (+0.37)	79.93 (−0.49)	**65.88** (+1.26)	55.90 (+3.26)	72.80 (+1.85)
GPT-OSS-120B	GEPA	80.83	79.17	97.24	79.36	91.00	71.80	83.23
	+ SA-Pareto	**82.50** (+1.67)	81.67 (+2.50)	**98.24** (+1.00)	**79.80** (+0.44)	**92.88** (+1.88)	**73.12** (+1.32)	**84.70** (+1.47)
	+ SA-Proposer (SAGE)	80.83	**83.33** (+4.16)	97.86 (+0.62)	79.30 (−0.06)	91.75 (+0.75)	72.55 (+0.75)	84.27 (+1.04)

Table 2 adds the two components one at a time: GEPA, then parent selection with the self-attribution front (§3.1), then the self-attribution proposer (§3.2), which gives full SAGE.

The two components contribute in different regimes. Selection alone raises GEPA's average on every model ($+0.62$, $+1.76$, $+2.71$, $+1.47$), and on the two larger models it gives the best average of all three rows. The proposer adds the most where selection leaves the most room. On Qwen3-4B, selection alone still loses to GEPA on four of the six benchmarks, and only with the proposer does SAGE beat GEPA on all six.

Each component helps where its mechanism predicts. The proposer targets mostly correct parents whose mutations would otherwise only rephrase them, so it should help most on unsaturated tasks that admit many strategies. Sokoban on Qwen3-4B fits this pattern: the gain over GEPA rises from $+5.12$ to $+10.75$ when the proposer is added. On the two larger models, where most tasks are close to saturation, the proposer lowers the average slightly (-0.86 and -0.43), but SAGE stays above GEPA; the gate admits a child only if it keeps full minibatch accuracy, which limits the damage. Selection thus gives a reliable gain on every model, and the proposer adds the largest gains where accuracy-only search stalls. Appendix G examines the mechanism directly by replaying every parent-selection decision in all 20 optimization runs: a rescued parent lies on the ancestry of the returned prompt in 12 runs, and a child of the self-attribution recipe in 8. Appendix H replaces the margin in parent selection with a diversity signal that does not use likelihoods.

4.4 Robustness: distribution shift, transfer, and black-box deployment

The results in Table 1 are in-distribution. We now test SAGE in three harder settings: generalization to harder problems, transfer of prompts across models, and a closed-source executor whose self-attribution is computed by an open-weight model. SAGE outperforms GEPA in all three.

Out-of-distribution difficulty. A prompt tuned on easy instances may simply encode the structure of those instances. We take the prompts that each method optimized on 2-box Sokoban and evaluate them, without re-optimization, on a harder 3-box test split. Accuracy drops for every method, so the question is whether the advantage survives. It does: SAGE gives the best prompt on all four models (Table 3) and beats GEPA on each by 1 to 2 points. Unlike the in-distribution average, this margin does not shrink with model size, which suggests that SAGE's prompts are less tied to the training difficulty.

Table 3: **Out-of-distribution difficulty (Sokoban).** Held-out accuracy (%, Average@4) of the Zero-shot CoT, GEPA, and SAGE prompts, each optimized on 2-box boards and evaluated *without re-optimization* on a harder 3-box test split (6×6 grid). Values in parentheses are improvements (percentage points) over Zero-shot CoT; bold marks the best method per model.

Method	Qwen3-4B	GPT-OSS-20B	Qwen3-30B-A3B	GPT-OSS-120B
Zero-shot CoT	21.50	48.00	29.00	69.50
GEPA	22.50 (+1.00)	51.00 (+3.00)	32.50 (+3.50)	71.50 (+2.00)
SAGE	**24.50** (+3.00)	**52.00** (+4.00)	**34.00** (+5.00)	**73.00** (+3.50)

Cross-model prompt transfer. A prompt's advantage could be specific to the model on which it was optimized. To test this, we transfer the WikiSQL prompts between Qwen3-30B-A3B and GPT-OSS-120B in both directions, without re-optimization (Table 4). SAGE's prompts transfer better in both directions. The difference is largest when moving to the stronger model: on GPT-OSS-120B, the prompt that SAGE optimized on Qwen3-30B-A3B scores 5.0 points above the corresponding GEPA prompt (60.10 vs. 55.10), whereas the two optimizers differ by less than one point when run on GPT-OSS-120B itself (72.55 vs. 71.80). SAGE's transfer advantage is thus largest on the model where its in-distribution advantage is smallest, which argues against reading the shrinking in-distribution gains as a sign that the signal matters less at scale.

Table 4: **Cross-model prompt transfer (WikiSQL).** A prompt optimized on the source model is evaluated on the target model without re-optimization (held-out accuracy, %, Average@4). *Target Zero-shot CoT* is the target model's own seed prompt, re-measured on the same server; values in parentheses are improvements (percentage points) over it, and bold marks the better transferred prompt.

Transfer	Target Zero-shot CoT	GEPA	SAGE
Qwen3-30B → GPT-OSS-120B	51.30	55.10 (+3.80)	**60.10** (+8.80)
GPT-OSS-120B → Qwen3-30B	52.10	67.70 (+15.60)	**68.60** (+16.50)

Closed-source model with a surrogate probe. The probe needs teacher-forced log-probabilities, which most commercial APIs do not provide. To test SAGE in this setting, we use Claude Haiku 4.5, accessed through the AWS Bedrock Converse API, as both the task executor and the reflection model. Because the API returns no log-probabilities, we compute the self-attribution margin with an open-weight *surrogate*, Qwen3-30B-A3B, which belongs to a different model family. SAGE outperforms GEPA on three of the four test sets and on average (47.33 vs. 46.11; Table 5), and its largest gain is on the hardest task, Sokoban (41.12 vs. 36.62, +4.50 points). The probe therefore does not have to be the executor: a different model can judge whether an instruction explains a trajectory better than its competitors do. SAGE does not require white-box access to the task model. The complete GEPA and SAGE study on this model cost about $119 in API fees.

Table 5: **Closed-source model.** Claude Haiku 4.5 through the AWS Bedrock Converse API (held-out test accuracy, %, Average@4). The task executor and reflection model are a black box; the self-attribution margin is computed by an open-weight *surrogate* (Qwen3-30B-A3B, teacher-forced scoring), separate from the task model. Values in parentheses are improvements (percentage points) over Zero-shot CoT; bold marks the best method in each column.

Method	AIME'25	AIME'26	Sokoban	WikiSQL	Average
Zero-shot CoT	34.17	33.33	26.38	51.19	36.27
GEPA	33.33 (−0.84)	**43.33** (+10.00)	36.62 (+10.24)	71.17 (+19.98)	46.11 (+9.84)
SAGE	**35.83** (+1.66)	40.83 (+7.50)	**41.12** (+14.74)	**71.55** (+20.36)	**47.33** (+11.06)

5 Related Work

Prompt optimization. Prompt optimizers search over instructions for a frozen model using templated mutation (Zhou et al., 2023; Guo et al., 2024; Fernando et al., 2023), an LLM that improves on a history of scored prompts (Yang et al., 2024), surrogate search over instructions and demonstrations (Opsahl-Ong et al., 2024; Khattab et al., 2024; Agarwal et al., 2025), or reflection on execution traces (Pryzant et al., 2023; Wang et al., 2024a; Yuksekgonul et al., 2024; Agrawal et al., 2026; Zhang et al., 2026b). The same accuracy-driven selection carries over as the approach extends from instructions to agent designs, algorithms, and self-modifying code (Hu et al., 2025; Cheng et al.,

2024; Zhang et al., 2025; Novikov et al., 2025; Zhang et al., 2026a; Lange et al., 2026). The closest existing idea to a second signal is GEPA's instance-level Pareto front, which keeps a candidate that is best on some individual problem even when its average is not. Being best on a problem is still a property of the reward matrix, however, so Corollary J.7 applies: an instance-level front inherits the blind spot of the average it refines. SAGE measures identifiability from the trajectories instead, and adding this front alone raises GEPA's average on all four models (§4.3).

Does the instruction cause the behavior? Several studies find that instructions contribute less to correct behavior than their wording suggests. Irrelevant or even misleading instructions can perform as well as sound ones (Webson & Pavlick, 2022), demonstrations help even when their labels are wrong (Min et al., 2022), meaning-preserving reformatting changes accuracy by tens of points (Sclar et al., 2024; Salinas & Morstatter, 2024; He et al., 2024; Errica et al., 2025; Mizrahi et al., 2024), and a model's stated reasoning can omit the factor that actually determined its answer (Turpin et al., 2023; Chen et al., 2025; Arcuschin et al., 2025). Together, these results show that a high score does not certify that the instruction did the work, which is the phenomenon that §2.2 formalizes. They establish it after the fact and in aggregate, however, and provide no per-candidate quantity that a search could use. We turn the question into a per-candidate, per-trajectory measurement that is cheap enough to run inside the optimization loop, so that it can guide selection instead of only auditing the result.

Attributing behavior to a context. Our probe addresses the same inference problem as prompt inversion, which recovers a hidden instruction from the outputs produced under it (Morris et al., 2024; Zhang et al., 2024; Sha & Zhang, 2024; Nazir et al., 2025). Context attribution asks which parts of a given context are responsible for a generation (Cohen-Wang et al., 2024), and influence functions ask the same question about training data (Grosse et al., 2023). These methods analyze one candidate at a time after the fact, and inversion treats identifiability as an attacker's goal. We use the same posterior over instructions, computed with a standard variational bound on mutual information (Barber & Agakov, 2003; Poole et al., 2019) with the frozen model's likelihood as the critic, but for the opposite purpose: Theorem J.13 turns how strongly this posterior concentrates on the true instruction into a lower bound on the information that reward discards. Identifiability thus becomes a selection criterion during the search instead of an audit afterwards.

6 Conclusion

Prompt evolution has relied on accuracy as its only selection signal. We showed that this leaves a blind spot: among equally scoring candidates, which a successful search increasingly produces, accuracy cannot separate an instruction with its own strategy from a free rider. We introduced self-attribution, a second signal that lower-bounds the information accuracy discards; showed that it should be used as an accuracy-gated constraint and not as a reward; and built SAGE, an extension of a state-of-the-art optimizer that adds one gradient-free measurement per trajectory. Across four models and six tasks, SAGE achieves the highest average accuracy on every model, rarely falls below its starting prompt, and keeps its advantage out of distribution, under cross-model transfer, and with a black-box API model. More broadly, our results suggest that the fitness signal, and not only the search procedure, leaves room to improve prompt optimization: measuring whether a candidate earned its reward adds robustness that optimizing accuracy harder does not.

7 Limitations

Under the fixed budget of 1800 metric calls, we prioritized breadth (four models, six tasks, and three robustness studies) over repeated runs, so each (model, task) cell comes from a single optimization run. Repeating every cell with several seeds would multiply the cost of the whole 24-cell grid, and of the robustness studies, by the number of seeds. We instead quantify uncertainty over test instances with a paired bootstrap (Appendix E.2) and report variance across three optimization seeds only for Qwen3-4B (Appendix E.1). Our theory assumes a frozen model and a reward that is a deterministic function of the trajectory, so it does not cover stochastic graders. Finally, the probe needs log-probabilities from some open-weight model; a surrogate worked for the one closed-source executor we tested, but we have not tested other pairings.

AI USE STATEMENT

We used large language models (LLMs) to polish the writing of this paper and to help find related work. All related works found by AI are verified manually by human. The authors reviewed all LLM-assisted text and references and take full responsibility for the content of the paper.

REFERENCES

Eshaan Agarwal, Raghav Magazine, Joykirat Singh, Vivek Dani, Tanuja Ganu, and Akshay Nambi. PromptWizard: Optimizing prompts via task-aware, feedback-driven self-evolution. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19974–20003, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1025. URL https://aclanthology.org/2025.findings-acl.1025/.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=L8094Whth0.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2026. URL https://arxiv.org/abs/2505.23281.

David Barber and Felix Agakov. Information maximization in noisy channels: A variational approach. In S. Thrun, L. Saul, and B. Schölkopf (eds.), *Advances in Neural Information Processing Systems*, volume 16. MIT Press, 2003. URL https://proceedings.neurips.cc/paper_files/paper/2003/file/a6ea8471c120fe8cc35a2954c9b9c595-Paper.pdf.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, Vlad Mikulik, Samuel R. Bowman, Jan Leike, Jared Kaplan, and Ethan Perez. Reasoning models don't always say what they think, 2025. URL https://arxiv.org/abs/2505.05410.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms, 2024. URL https://arxiv.org/abs/2406.16218.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Madry. Contextcite: Attributing model generation to context. In *ICML 2024 Next Generation of AI Safety Workshop*, 2024. URL https://openreview.net/forum?id=U4d4BIhwSt.

Federico Errica, Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. What did I do wrong? quantifying LLMs' sensitivity and consistency to prompt engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*

(Volume 1: Long Papers), pp. 1543–1558, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.73. URL https://aclanthology.org/2025.naacl-long.73/.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution, 2023. URL https://arxiv.org/abs/2309.16797.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions, 2023. URL https://arxiv.org/abs/2308.03296.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. Does prompt formatting have any impact on llm performance?, 2024. URL https://arxiv.org/abs/2411.10541.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work?, 2022. URL https://arxiv.org/abs/2202.12837.

Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. State of what art? a call for multi-prompt LLM evaluation. *Transactions of the Association for Computational Linguistics*, 12:933–949, 2024. doi: 10.1162/tacl_a_00681. URL https://aclanthology.org/2024.tacl-1.52/.

John Xavier Morris, Wenting Zhao, Justin T Chiu, Vitaly Shmatikov, and Alexander M Rush. Language model inversion. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=t9dWHpGkPj.

Murtaza Nazir, Matthew Finlayson, John X. Morris, Xiang Ren, and Swabha Swayamdipta. Better language model inversion by compactly representing next-token distributions, 2025. URL https://arxiv.org/abs/2506.17090.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo

Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.525. URL https://aclanthology.org/2024.emnlp-main.525/.

Ben Poole, Sherjil Ozair, Aaron van den Oord, Alexander A. Alemi, and George Tucker. On variational bounds of mutual information, 2019. URL https://arxiv.org/abs/1905.06922.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Abel Salinas and Fred Morstatter. The butterfly effect of altering prompts: How small changes and jailbreaks affect large language model performance. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4629–4651, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.275. URL https://aclanthology.org/2024.findings-acl.275/.

Max-Philipp B. Schrader. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. The prompt report: A systematic survey of prompt engineering techniques, 2025. URL https://arxiv.org/abs/2406.06608.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=RIu5lyNXjT.

Zeyang Sha and Yang Zhang. Prompt stealing attacks against large language models, 2024. URL https://arxiv.org/abs/2402.12959.

Joar Max Viktor Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yb3HOXO3lX2.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=bzs4uPLXvi.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=22pyNMuIoa.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024b. URL https://openreview.net/forum?id=y10DM6R2r3.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2504.20073.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2300–2344, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.167. URL https://aclanthology.org/2022.naacl-main.167/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text, 2024. URL https://arxiv.org/abs/2406.07496.

Collin Zhang, John Xavier Morris, and Vitaly Shmatikov. Extracting prompts by inverting LLM outputs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14753–14777, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.819. URL https://aclanthology.org/2024.emnlp-main.819/.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2025. URL https://arxiv.org/abs/2410.10762.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026b. URL https://arxiv.org/abs/2510.04618.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=uccHPGDlao.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning, 2018. URL https://openreview.net/forum?id=Syx6bz-Ab.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

APPENDIX CONTENTS

A Experimental Setup **17**
A.1 Datasets and Train/Val/Test Splits 17
A.2 Models and Inference Parameters 18
A.3 Optimization Hyperparameters 18
A.4 Hardware 19

B SAGE Pseudo-Code **19**

C Cost of the Self-Attribution Probe **19**

D Search Effort: Reflection Calls and Candidate-Pool Size **20**

E Statistical Significance **21**
E.1 Variance Across Optimization Seeds 22
E.2 Paired Instance Bootstrap 22

F Best-Prompt Length **24**

G Mechanism Analysis: Which Component Produces the Gain **24**
G.1 Selection: Parents the Accuracy Fronts Would Drop 24
G.2 Variation: Self-Attribution Children on the Winning Ancestry 26

H Self-Attribution Versus Generic Diversity **26**

I Validation Curves **27**

J Why Self-Attribution Is Informative **29**
J.1 Setup and Notation 29
J.2 Reward Carries No Information Where the Trajectory Does 30
J.3 The Self-Attribution Margin Lower-Bounds This Information 33
J.4 A High Score Does Not Mean a Good Prompt 35
J.5 Summary and Scope 36

K Reflection Meta-Prompts **37**

L Best Prompts for Every Benchmark **38**
L.1 AIME'25 38
L.2 MATH500 49
L.3 MMLU-Pro 53
L.4 Sokoban 60
L.5 WikiSQL 73

A Experimental Setup

A.1 Datasets and Train/Val/Test Splits

We evaluate on six tasks of four types: competition mathematics (MATH500, AIME'25, AIME'26), multi-domain multiple-choice reasoning (MMLU-Pro), symbolic planning (Sokoban), and text-to-SQL generation (WikiSQL). The tasks differ in answer format and scoring (boxed numeric answers, option letters, action plans, and executable queries) and range from single answers to multi-step plans.

- **MATH500** (Hendrycks et al., 2021; Lightman et al., 2023) is the 500-problem subset of MATH released with PRM800K (`HuggingFaceH4/MATH-500`). It contains competition problems from seven subjects and five difficulty levels. The model writes a step-by-step solution ending in a boxed answer, which is scored by symbolic equivalence. We stratify by difficulty level so that every split has the same mix of easy and hard problems. Split: 200 train / 100 val / 200 test.
- **MMLU-Pro** (Wang et al., 2024b) is a harder, ten-option successor to MMLU (`TIGER-Lab/MMLU-Pro`) that covers 14 academic and professional domains, such as law, engineering, health, and economics. The model answers each multiple-choice question with chain-of-thought reasoning and is scored by exact match of the option letter. This task tests whether the method helps outside mathematics. We subsample from the pool of about 12k questions, stratified by category so that all 14 domains appear in each split. Split: 400 train / 150 val / 400 test.
- **WikiSQL** (Zhong et al., 2018) is a single-turn text-to-SQL task (`Salesforce/wikisql`). Given a question and a table schema with a few sample rows, the model writes one SQL query, which is scored by *execution match*: the query is run on the table in memory and its result is compared with that of the gold query. We use a *table-disjoint* split, so each table (schema and rows) appears in only one split and no test table is seen during optimization. A test gain therefore reflects generalization to unseen schemas, not memorized columns. We subsample from the pool of about 15k questions. Split: 400 train / 150 val / 400 test.
- **AIME'25 / AIME'26** are the 2025 and 2026 American Invitational Mathematics Examinations, each with 30 hard olympiad problems with integer answers. Train and val problems come from the 2022–2024 contests (`AI-MO/aimo-validation-aime`, 90 problems, split 45/45). Each test set is a complete, unseen contest: `MathArena/aime_2025` or `MathArena/aime_2026` (Balunović et al., 2026). AIME'26 took place after the training cutoff of every model we use, so a gain on it cannot come from memorized solutions; it serves as our contamination-resistant test set.
- **Sokoban** (Wang et al., 2025; Schrader, 2018) is a grid puzzle in which the agent must push every box onto a target cell without creating a deadlock. We use a single-shot formulation: the model reads one board, given as an ASCII grid with explicit coordinates, and outputs the *entire* action plan at once. The plan is scored by replaying it in a fresh environment and checking that all boxes end on targets. Boards are generated procedurally from three disjoint seed ranges, so train, val, and test boards are distinct but come from the same distribution, and they are identical across processes and models. The training distribution has 2 boxes on a 6×6 grid; split: 200 train / 100 val / 200 test. The out-of-distribution study (§4.4) evaluates the prompts optimized on 2-box boards, without re-optimization, on a harder test split of 50 boards with 3 boxes.

Train/val/test protocol. SAGE optimizes prompts for a *frozen* model, so the three splits play different roles than in weight training. The **train** split is the only data the search optimizes against: minibatches are drawn from it to evaluate a parent, build the reflection examples, and evaluate a proposed child. The **val** split is the optimizer's internal benchmark. It defines the accuracy Pareto fronts, every accepted candidate is evaluated on it in full, and the returned prompt is the candidate with the highest validation accuracy. Because of these full evaluations, the val split accounts for most of the cost, which is why we keep it small. The **test** split is used once, after optimization, to compute Accuracy Average@4; the search, the fronts, and the selection rule never see it. Every reported number

therefore measures a prompt chosen without test feedback, and the gap between validation and test accuracy is a fair measure of overfitting. For the single-pool datasets (MATH500, MMLU-Pro, WikiSQL), the three splits come from one stratified shuffle with a fixed seed (`DATA_SPLIT_SEED` $= 17$), so the partition is reproducible and each split keeps the pool's mix of difficulty levels or domains; WikiSQL additionally keeps tables disjoint across splits. For AIME and Sokoban, the split follows from the data (a separate held-out contest and disjoint generator seed ranges, respectively), which separates train and test more strictly than a random partition of one pool.

A.2 Models and Inference Parameters

We use four open-weight models, listed by parameter count, each serving as both the task executor and the reflection model. All are served locally with vLLM (Kwon et al., 2023). The self-attribution probe reads teacher-forced log-probabilities from the same server through a `/completions` call with `echo=true`, `max_tokens=0`, and `logprobs=0`, i.e., prefill only, with no decoding. All four models use the same sampling parameters, and evaluation always uses Accuracy Average@4 (accuracy averaged over four samples per test problem). The per-model settings are:

- **Qwen3-4B-Instruct-2507** (Yang et al., 2025): temperature 0.7, $\text{top}_p = 0.8$, $\text{top}_k = 20$, presence penalty 1.0; `max_tokens` $= 16384$ for generation and 4096 for reflection.
- **Qwen3-30B-A3B-Instruct-2507** (Yang et al., 2025): a mixture-of-experts model (30B total / 3B active parameters) with the same sampling parameters as the 4B model (temperature 0.7, $\text{top}_p = 0.8$, $\text{top}_k = 20$, presence penalty 1.0); `max_tokens` $= 16384$ for generation and 4096 for reflection.
- **GPT-OSS-20B** (OpenAI et al., 2025): a reasoning model run with `reasoning_effort=high` and the same sampling parameters as the Qwen models (temperature 0.7, $\text{top}_p = 0.8$, $\text{top}_k = 20$, presence penalty 1.0). The reasoning trace (`reasoning_content`) and the final answer (`content`) are returned separately and concatenated for evaluation. The generation budget is 32768 tokens, shared with the reasoning trace; reflection uses `max_tokens` $= 65536$.
- **GPT-OSS-120B** (OpenAI et al., 2025): a reasoning model (117B total / 5B active parameters) run with `reasoning_effort=medium`, the same sampling parameters, and the same handling of the reasoning trace as the 20B model. The generation budget is 32768 tokens (or the full remaining context window); reflection uses `max_tokens` $= 65536$.

In the closed-source study (§4.4), the executor and reflection model are Claude Haiku 4.5 on AWS Bedrock (Converse API, a black box with no log-probabilities; temperature 0.7; extended thinking disabled), and the probe runs on a Qwen3-30B-A3B surrogate.

A.3 Optimization Hyperparameters

All methods (Zero-shot CoT, Few-shot CoT, MIPROv2, GEPA, SAGE, and the ablations) share one protocol, so all comparisons are at matched budget. SAGE keeps the objective, Pareto machinery, and final-selection rule of GEPA (Agrawal et al., 2026) and adds only the self-attribution hyperparameters below.

- **Rollout budget:** 1800 metric calls for GEPA and SAGE, whose realized totals lie in $[1805, 1940]$. MIPROv2 is calibrated per dataset to the same interval (see below). Zero-shot CoT and Few-shot CoT perform no search and use only the final evaluation.
- **Minibatch size:** $b = 5$ problems per iteration, reshuffled every epoch.
- **Evaluation:** Accuracy Average@4 on the held-out test split, i.e., accuracy averaged over four samples per test problem.
- **Margin threshold:** $\tau = 0$. On the all-correct path, the acceptance gate keeps a child if its mean margin over the minibatch exceeds τ.
- **Proposer routing:** the self-attribution recipe is used when the parent's minibatch failure rate is below $\theta = 0.4$ (*mostly correct*), its self-attribution score is at or below the population median, and it has an attributional nearest neighbor to contrast with. Otherwise, the host's failure recipe is used, as in GEPA.

Few-shot CoT. We collect the examples separately for each (model, dataset) pair. We query the model on the train split with the same seed instruction, sampling parameters, template, and scorer as everywhere else, go through the split in a fixed shuffled order, and keep the first $K = 3$ responses that score at least 0.5. Each example keeps its full reasoning trace, except on WikiSQL, which is scored on the final query alone and keeps only that query. Two safeguards keep the examples from becoming a confound. First, each example is limited to 8000 characters; a candidate that is too long is skipped, not truncated, so the three examples together take at most about 6K tokens of a 131K context and cannot crowd out the answer. Second, on the multiple-choice task, we replace one example if all three share the same correct option letter. AIME's train split has only 45 problems, so if the quota cannot be filled, we first relax the length limit and then fall back to best-of-4 sampling. The examples are inserted into the seed instruction as a single system prompt and evaluated with the same harness as every other method.

MIPROv2. Matching MIPROv2's budget requires care, because its total number of metric calls is not a hyperparameter but follows from other settings:

$$\text{total} \;=\; \underbrace{b_{\mathrm{M}}\,T}_{\text{minibatch trials}} \;+\; \underbrace{n_{\mathrm{val}}\big(\lceil T/10\rceil + 1\big)}_{\text{periodic full validation}} \;+\; \underbrace{c_{\mathrm{boot}}}_{\text{bootstrapping}}\;, \tag{3}$$

where b_{M} is the minibatch size, T the number of trials, and n_{val} the size of the validation set. This formula reproduces the observed totals exactly. Because n_{val} differs across datasets, no single choice of (T, b_{M}) puts every dataset in range, so we calibrate these two settings per dataset (Table 6) to land in $[1805, 1940]$. The remaining settings are chosen to be comparable to GEPA and SAGE: at most three bootstrapped demonstrations and no labeled ones, ten instruction candidates, and the model under test as the instruction-proposal model. The returned instruction and demonstrations are rendered in the same way as for Few-shot CoT and scored with the same harness.

Table 6: **MIPROv2 budget calibration.** Per-dataset values of the two settings in Eq. 3, chosen so that the realized number of metric calls matches that of GEPA and SAGE. What is held fixed across methods is the total budget, not these settings, which have no counterpart in GEPA or SAGE. The two datasets with $n_{\mathrm{val}} = 150$ need a minibatch size of 27: with 25, no number of trials lands in the target range, because the full validation pass every tenth trial makes the total jump over it.

Dataset	n_{val}	Trials	Minibatch	Metric calls
AIME	45	60	25	1835
Sokoban	100	50	25	1880
MATH500	100	50	25	1879
WikiSQL	150	40	27	1872
MMLU-Pro	150	40	27	1862

A.4 Hardware

All experiments were run on a single node with 8 NVIDIA H100 GPUs (80 GB each). Each open-weight model is served locally with vLLM on this node.

B SAGE Pseudo-Code

Algorithms 1–3 give the full SAGE procedure. Only the two subroutines SelectParent (§3.1) and ProposeAndAccept (§3.2) differ from the host.

Here the margin $m_j^{\mathrm{tot}}(x; C) = \ell^{\mathrm{tot}}(z_j; \Phi_j) - \max_{\Phi_k \in C,\, k \neq j} \ell^{\mathrm{tot}}(z_j; \Phi_k)$ is computed from the cross-score matrix $L[j, k] = \log \pi(z_j \mid \mathrm{chat}(x, \Phi_k))$, restricted to the candidates in C (§2.3).

C Cost of the Self-Attribution Probe

The only computation SAGE adds to GEPA is the self-attribution probe, which scores stored (trajectory, instruction) pairs by teacher forcing. To isolate its cost, we compare the + SA-Pareto

Algorithm 1 SAGE (Self-Attribution-Guided Evolving)

Require: frozen model π; seed instruction Φ_0; train $\mathcal{D}$; val $\mathcal{V}$; budget B; minibatch size b; margin floor τ; failure-rate cut θ
Ensure: returned instruction $\Phi^\star$

$\mathcal{P} \leftarrow \{\Phi_0\}$; evaluate Φ_0 on $\mathcal{V}$; used $\leftarrow |\mathcal{V}|$
while used $< B$ **do**
  $\Phi_p \leftarrow$ SELECTPARENT$(\mathcal{P}, \mathcal{V})$ ▷ Alg. 2
  $M \leftarrow$ minibatch of b problems from $\mathcal{D}$
  $\Phi_c \leftarrow$ PROPOSEANDACCEPT$(\Phi_p, M, \mathcal{P})$ ▷ Alg. 3
  if $\Phi_c \neq \perp$ **then**
    evaluate Φ_c on $\mathcal{V}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{\Phi_c\}$; used $+= |\mathcal{V}| + b$
  else
    used $+= b$
  end if
end while
return $\Phi^\star \leftarrow \arg\max_{\Phi \in \mathcal{P}} \mathrm{acc}(\Phi, \mathcal{V})$

Algorithm 2 SELECTPARENT: parent selection with a self-attribution front (§3.1)

Require: pool $\mathcal{P}$; val set $\mathcal{V}$
Ensure: parent Φ_p

for all $v \in \mathcal{V}$ **do**
  $F_v^{\mathrm{acc}} \leftarrow \arg\max_{\Phi \in \mathcal{P}} \mathrm{acc}(\Phi, v)$ ▷ host: per-instance accuracy front
  $C_v \leftarrow \{\Phi \in \mathcal{P} : \mathrm{acc}(\Phi, v) = 1\}$ ▷ candidates correct on v
  if $|C_v| \geq 2$ **then**
    $F_v^{\mathrm{sa}} \leftarrow \{\Phi_j \in C_v : m_j^{\mathrm{tot}}(v; C_v) > 0\}$ ▷ self-attribution front, §2.3
  else
    $F_v^{\mathrm{sa}} \leftarrow \varnothing$
  end if
end for
$F \leftarrow \bigcup_{v \in \mathcal{V}} \left(F_v^{\mathrm{acc}} \cup F_v^{\mathrm{sa}}\right)$
prune every $\Phi \in F$ that is dominated on every v ▷ host's own pruning, unchanged
sample $\Phi_p \in F$ with frequency-weighted probability ▷ host's own sampler, unchanged
return Φ_p

ablation of Table 2 with GEPA, since the two differ *only* in the probe: they use the same proposer, skip all-correct minibatches in the same way, and have the same budget of about 1800 metric calls. Two properties bound the cost of the probe. Every probe token is *prefill-only* (scored with `max_tokens` $= 0$, so no token is decoded), and each pair is scored *only once*, since a persistent cache remains valid for a frozen model.

A large overhead in FLOPs translates into a small overhead in wall-clock time (Table 7). A prefill token and a decode token of the same model cost roughly the same number of FLOPs, so within a model the FLOPs overhead of the probe reduces to a ratio of token counts. Weighted by tokens across all runs, the probe processes about $4\times$ as many tokens as generation does. Yet it adds only about 5% to end-to-end wall-clock time ($286 \rightarrow 302$ minutes on the same server under the same budget). The two kinds of work have different bottlenecks. *Decoding* is memory-bandwidth-bound and sequential: each generated token requires a full forward pass that streams the weights from HBM, and token $t{+}1$ must wait for token t. The *probe* is compute-bound and parallel: a single prefill pass scores a whole teacher-forced sequence, amortizing the weight streaming over thousands of tokens, and vLLM's continuous batching packs many concurrent `max_tokens` $= 0$ requests together. A $4\times$ overhead in prefill FLOPs therefore costs only about 5% in wall-clock time.

D SEARCH EFFORT: REFLECTION CALLS AND CANDIDATE-POOL SIZE

Table 8 reports the number of reflection-model calls and the final candidate-pool size of GEPA and SAGE for every (model, task) pair. Under the same budget of 1800 metric calls, the two methods

Algorithm 3 PROPOSEANDACCEPT: self-attribution proposer and all-correct acceptance gate (§3.2)

Require: parent Φ_p; minibatch M; pool $\mathcal{P}$
Ensure: accepted child Φ_c, or $\perp$ if rejected

roll out Φ_p on M; $f \leftarrow 1 - \mathrm{acc}(\Phi_p, M)$ ▷ failure rate on minibatch
$\mathrm{nn} \leftarrow$ attributional nearest neighbor of Φ_p in $\mathcal{P}$
if $f < \theta$ **and** $\mathrm{score}(\Phi_p) \leq \mathrm{median}_{\Phi \in \mathcal{P}} \mathrm{score}(\Phi)$ **and** $\mathrm{nn} \neq \perp$ **then**
    $\Phi_c \leftarrow$ ATTRIBUTIONRECIPE(Φ_p, nn, M) ▷ SAGE recipe (§K)
else
    $\Phi_c \leftarrow$ FAILURERECIPE(Φ_p, M) ▷ host recipe, unchanged
end if
roll out Φ_c on M
if $\mathrm{acc}(\Phi_p, M) = 1$ **then** ▷ all-correct path: SA acceptance gate
    $\bar{m} \leftarrow \frac{1}{|M|} \sum_{x \in M} m_c^{\mathrm{tot}}\big(x; \{\Phi_p, \mathrm{nn}\}\big)$
    if $\mathrm{acc}(\Phi_c, M) = 1$ **and** $\bar{m} > \tau$ **then return** Φ_c ▷ accuracy holds first, then margin
    end if
else ▷ host path: strict accuracy improvement
    if $\mathrm{acc}(\Phi_c, M) > \mathrm{acc}(\Phi_p, M)$ **then return** Φ_c
    end if
end if
return $\perp$

Table 7: **Cost of the self-attribution probe.** Per model (mean over the five tasks): number of probe calls, probe tokens (prefill-only, teacher-forced), generation tokens of the run as a reference, and the resulting FLOPs overhead (probe tokens divided by generation tokens; a prefill token and a decode token cost about the same FLOPs, so within a model the FLOPs ratio reduces to a token ratio). Every probe token is prefill-only (`max_tokens` $= 0$, no decoding), and each (trajectory, instruction) pair is scored only once (persistent cache; the model is frozen). The last row is the *wall-clock* cost of adding the probe to GEPA on the same server under the same budget (mean over the measured (model, task) runs): a roughly $4\times$ overhead in prefill tokens costs only about 5% in wall-clock time, because prefill is compute-bound and batched, whereas decoding is memory-bandwidth-bound and sequential.

Model	Probe calls	Probe tokens (prefill-only)	Gen tokens (reference)	FLOPs overhead
Qwen3-4B	5,059	16.1M	7.0M	2.3×
Qwen3-30B-A3B	7,165	28.4M	7.4M	3.8×
GPT-OSS-20B	8,815	56.9M	13.7M	4.2×
GPT-OSS-120B	10,366	35.5M	6.4M	5.5×
Token-weighted mean FLOPs overhead (all 20 runs)				4.0×
Wall-clock: GEPA 286 min → +probe 302 min				+5%

make a similar number of reflection calls and keep pools of similar size, except on MATH500, where SAGE keeps more candidates (16 in every run, versus 6 to 12 for GEPA). SAGE's gains therefore come mainly from *which* candidates survive, not from generating or keeping more of them.

E STATISTICAL SIGNIFICANCE

We check the SAGE − GEPA gap against two sources of noise. The first is the randomness of the search itself: run again, an optimizer may return a different prompt (§E.1). The second is the sampling of test problems: a fixed pair of prompts may differ on this test set by chance (§E.2).

Table 8: **Search effort: reflection-model calls (R) and final candidate-pool size (P)** for GEPA and SAGE. Under the same budget of 1800 metric calls, the two methods make a similar number of reflection calls and keep pools of similar size, except on MATH500, where SAGE keeps more candidates. The self-attribution signal mainly changes *which* candidates survive, not how many are generated or kept.

	Reflection-LM calls (R)				**Candidate pool (P)**			
	Qwen3-4B		GPT-OSS-20B		Qwen3-4B		GPT-OSS-20B	
Task	GEPA	SAGE	GEPA	SAGE	GEPA	SAGE	GEPA	SAGE
AIME'25	84	72	66	78	21	24	22	22
MATH500	27	24	21	17	12	16	11	16
MMLU-Pro	26	27	25	22	10	11	11	11
Sokoban	42	37	24	30	14	15	16	15
WikiSQL	14	14	24	16	12	12	11	11
	Qwen3-30B-A3B		GPT-OSS-120B		Qwen3-30B-A3B		GPT-OSS-120B	
Task	GEPA	SAGE	GEPA	SAGE	GEPA	SAGE	GEPA	SAGE
AIME'25	65	65	63	79	24	26	25	23
MATH500	28	17	24	18	6	16	10	16
MMLU-Pro	23	18	30	17	10	11	10	11
Sokoban	31	24	42	27	15	16	13	16
WikiSQL	17	21	26	21	11	11	11	11

E.1 Variance Across Optimization Seeds

The prompt that an optimizer returns depends on the order of the minibatches and on the samples drawn from the executor and the reflection model, so a second run can return a different prompt. To measure this variance, we rerun GEPA and SAGE with two more seeds, using the same budget of 1800 metric calls and the same hyperparameters as in Table 1. The seed fixes the minibatch order and the sampling seeds of the vLLM server. We use seeds 0, 1, and 2; seed 0 is the run reported in Table 1.

We do this only for Qwen3-4B. Repeating the full grid would take 80 more optimization runs (20 runs for each of the two optimizers and two new seeds), each with 1800 metric calls, which is beyond our compute budget. Qwen3-4B needs 20 of these runs. We chose it because it has the largest SAGE − GEPA gap in Table 1, so it is the model on which that gap most needs to be checked. Zero-shot CoT involves no search, so its three runs differ only in test-time decoding and serve as a reference for decoding noise.

Table 9 reports the results. Averaged over the six benchmarks, SAGE reaches 62.81 ± 0.56, compared with 60.18 ± 0.80 for GEPA and 58.36 ± 0.39 for Zero-shot CoT. The per-seed gap between SAGE and GEPA is $+2.63 \pm 1.34$ points, and it is positive in all three seeds. This gap is more than three times the seed-to-seed standard deviation of either optimizer, so the advantage of SAGE on this model is not an artifact of one lucky run. It is smaller than the $+3.99$ of the single run in Table 1, which therefore overstates the gap somewhat.

SAGE is also less sensitive to the seed than GEPA. Its standard deviation is lower on all six benchmarks, and the difference is largest where GEPA varies most: on MATH500 (0.33 versus 1.94) and on Sokoban (1.15 versus 2.63). GEPA's mean over the three seeds is below the zero-shot seed on three benchmarks (AIME'26, MATH500, and Sokoban), whereas SAGE's mean is above it on all six. This matches the observation in §4.2 that accuracy-only search often ends below its starting prompt.

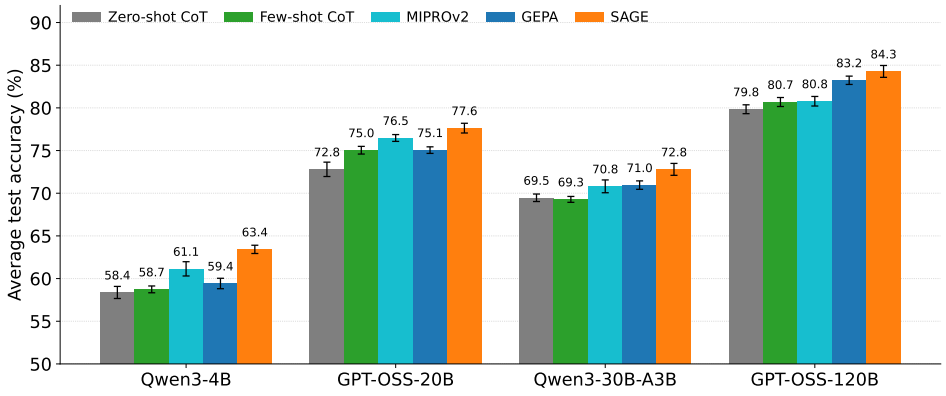
E.2 Paired Instance Bootstrap

Figure 3 shows the per-model averages as bar charts: **(a)** the main results of Table 1 for all five methods, and **(b)** the ablation of Table 2.

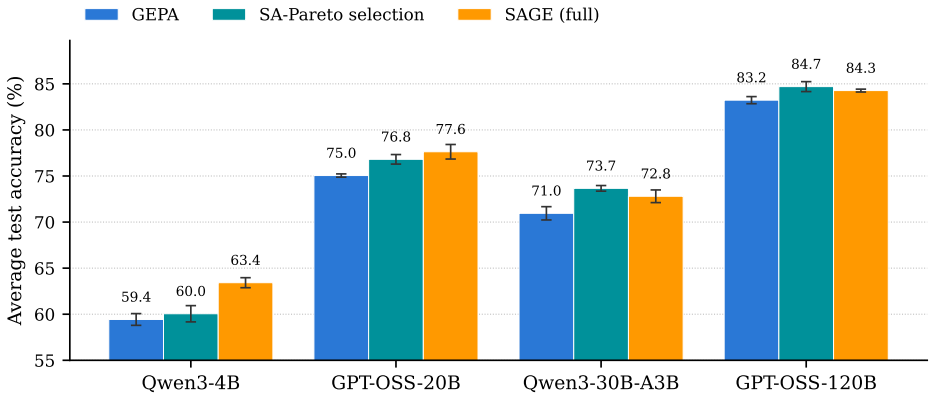
The error bars in Figure 3 reflect decoding noise, i.e., the spread of the average over the four sampling passes. On their own, they do not show that the SAGE − GEPA gap exceeds the noise from sampling test problems. Because they are *marginal* (one interval per bar), they also understate the

Table 9: **Variance across optimization seeds on Qwen3-4B** (held-out test Accuracy Average@4, %). Each optimizer is run three times with different seeds under the same budget of 1800 metric calls. The seeds are 0, 1, and 2, and seed 0 is the run reported in Table 1. Zero-shot CoT performs no search, so its three runs differ only in test-time decoding. Entries are mean ± sample standard deviation (with $n-1$ in the denominator) over the three seeds. The last column is the SAGE − GEPA difference, computed per seed and then averaged.

Benchmark	Zero-shot CoT	GEPA	SAGE	SAGE − GEPA
AIME'25	45.83 ± 1.67	46.94 ± 1.27	48.33 ± 0.84	+1.39 ± 1.92
AIME'26	52.22 ± 1.27	51.94 ± 1.27	54.17 ± 0.84	+2.22 ± 1.93
MATH500	92.20 ± 0.30	91.65 ± 1.94	94.55 ± 0.33	+2.90 ± 2.23
MMLU-Pro	63.99 ± 0.30	71.41 ± 1.12	72.18 ± 0.40	+0.77 ± 1.52
Sokoban	49.67 ± 1.01	49.58 ± 2.63	56.50 ± 1.15	+6.92 ± 3.75
WikiSQL	46.24 ± 0.35	49.54 ± 0.62	51.11 ± 0.45	+1.57 ± 0.40
Average	58.36 ± 0.39	60.18 ± 0.80	62.81 ± 0.56	+2.63 ± 1.34



(a) Main results: held-out test Accuracy Average@4 (%), averaged over the six tasks. SAGE has the highest average on every model.



(b) Ablation: GEPA, + SA-Pareto, and + SA-Proposer (full SAGE), added cumulatively. SA-Pareto selection improves on GEPA on every model but falls short of SAGE on the two smaller models, where the proposer closes the gap.

Figure 3: Per-model averages over all tasks. Error bars show ±1 standard error of the per-model mean over the four sampling passes of Accuracy Average@4, i.e., test-time decoding noise. Table 10 gives a paired analysis over test instances. **(a)** Main results; **(b)** ablation.

evidence: SAGE and GEPA are scored on the *same* test instances, so the relevant quantity is their paired difference, whose variance is reduced by the strong positive correlation between the two methods across problems (a problem that is easy for one method is usually easy for the other).

We therefore run a paired bootstrap of the average gap over test instances (Table 10). Each test instance is valued by its Average@4 score. A replicate resamples instances with replacement *within each task* (stratified, so the small 30-problem AIME test sets keep their weight), scores SAGE and GEPA on the *same* resampled instances, recomputes each task mean, and macro-averages the six task means, as in Table 1. Over $B{=}10^4$ replicates, the gap is significant on the three smaller models: $+4.00$ on Qwen3-4B (95% CI $[+1.83, +6.16]$, $p{=}0.0002$), $+2.58$ on GPT-OSS-20B ($[+0.39, +4.76]$, $p{=}0.011$), and $+1.84$ on Qwen3-30B-A3B ($[+0.14, +3.63]$, $p{=}0.017$). On these models, the advantage cannot be explained by the sampling of test problems, even with the small AIME test sets. On GPT-OSS-120B, the in-distribution gap of $+1.04$ is not significant ($[-0.78, +2.90]$, $p{=}0.13$). As noted in §4.2, our evidence for this model is instead the out-of-distribution and transfer results of §4.4, where the gap does not shrink with scale.

Table 10: **Paired instance bootstrap of the SAGE − GEPA gap** (held-out test Accuracy Average@4, %). Each row macro-averages over the six benchmarks. The difference is resampled *paired* over test instances ($B{=}10^4$, stratified within task), so the interval reflects the composition of the test set and not only decoding noise. Δ is SAGE − GEPA in percentage points; the CI is the paired 95% bootstrap interval; p is the one-sided bootstrap tail probability $\Pr(\Delta \le 0)$. The gap is significant at $p < 0.05$ on the three smaller models.

Model	GEPA	SAGE	Δ	Paired 95% CI	$p(\Delta \le 0)$
Qwen3-4B	59.42	63.42	$+4.00$	$[+1.83, +6.16]$	0.0002
GPT-OSS-20B	75.05	77.63	$+2.58$	$[+0.39, +4.76]$	0.011
Qwen3-30B-A3B	70.95	72.80	$+1.84$	$[+0.14, +3.63]$	0.017
GPT-OSS-120B	83.23	84.27	$+1.04$	$[-0.78, +2.90]$	0.131

F Best-Prompt Length

One might suspect that SAGE wins simply by writing longer, more detailed instructions. Figure 4 checks this: for every (model, task) pair, we count the tokens in the best prompt returned by each method and plot GEPA's length against SAGE's. Most points lie *below* the line $y = x$, so SAGE's prompt is usually shorter than GEPA's. The largest differences occur on the larger models, where GEPA's best prompts reach thousands of tokens while SAGE's remain compact; the main exception is Qwen3-4B on Sokoban, where SAGE's prompt is longer. SAGE's gains therefore do not come from longer instructions.

G Mechanism Analysis: Which Component Produces the Gain

The ablation in §4.3 shows *that* both components help on average. This appendix examines *how*, by reconstructing every decision that the two components affected in all 20 optimization runs. (There are four models and five training tasks; AIME is a single run evaluated on two test contests, which is why there are 20 runs but 24 benchmark cells in Table 1.) The reconstruction is exact: we reload the optimizer state and the cached probe log-probabilities from disk and call the host's own dominance pruning and weighted sampler, with no cache misses.

G.1 Selection: Parents the Accuracy Fronts Would Drop

Table 11 counts *rescues*: iterations at which the host's accuracy-only selection gives a candidate probability zero while the merged fronts give it a positive probability. Pruning by the accuracy fronts is the common case: it removes a candidate in 4007 of the 6022 candidate-iterations, and the self-attribution fronts return 533 of these to the pool of possible parents. Rescues occur in 15 of the 20 runs. In 12 runs, a rescued candidate lies on the ancestry of the returned prompt; in each of these,

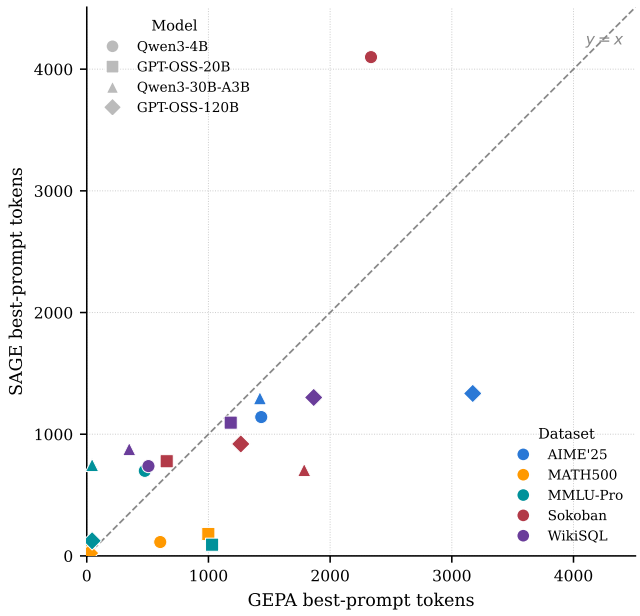


Figure 4: **Best-prompt length of GEPA and SAGE.** Each point is one (model, task) pair; the axes give the number of tokens in the best prompt returned by each method. Points below the line $y = x$ are pairs where SAGE's prompt is shorter than GEPA's.

Model	Task	#R	#R$_\ell$	#R$_p$	max P^ℓ_{m}	rescue → best
Qwen3-4B	AIME	3	2	2	33.0%	✓
	MATH500	4	2	2	29.4%	✓
	MMLU-Pro	0	0	0	–	–
	Sokoban	0	0	0	–	–
	WikiSQL	6	2	2	22.6%	✓
GPT-OSS -20B	AIME	7	2	7	31.5%	✓
	MATH500	7	2	3	29.8%	✓
	MMLU-Pro	1	1	0	–	✗
	Sokoban	0	0	0	–	–
	WikiSQL	0	0	0	–	–
Qwen3-30B -A3B	AIME	9	2	6	35.9%	✓
	MATH500	12	3	6	49.7%	✓
	MMLU-Pro	1	0	0	–	✗
	Sokoban	3	1	2	14.9%	✓
	WikiSQL	4	2	2	24.1%	✓
GPT-OSS -120B	AIME	2	2	2	36.8%	✓
	MATH500	7	0	6	–	✗
	MMLU-Pro	4	1	1	28.4%	✓
	Sokoban	0	0	0	–	–
	WikiSQL	5	2	4	48.4%	✓

Table 11: **Parents that the accuracy fronts drop, and what the self-attribution fronts do with them.** A *rescue* is an iteration at which a candidate's accuracy-only sampling probability is zero, so the host would drop it from the pool of possible parents, while the merged accuracy and self-attribution fronts give it $P_{\mathrm{m}} > 0$. **#R**: candidates rescued at least once; **#R$_\ell$**: rescued candidates on the ancestry of the returned prompt; **#R$_p$**: rescued candidates selected as a parent while rescued; **max** P^ℓ_{m}: the largest such probability on the ancestry. **rescue → best** is ✓ if some ancestor was rescued and selected as a parent while rescued, ✗ if rescues occurred but no ancestor meets this condition, and – if the run has no rescue. Discussed in §G.1.

the candidate was selected as a parent while rescued, not merely kept in the pool, and the largest sampling probability of a rescued ancestor ranges from 15% to 50% across these runs.

G.2 Variation: Self-Attribution Children on the Winning Ancestry

		SAGE				GEPA (host)		
Model	**Task**	**ancestry (seed → best)**	**depth**	**val**	**test**	**depth**	**val**	**test**
Qwen3-4B	AIME	seed → F → F → F	3	0.689	49.2	2	0.711	46.7
	MATH500	seed → F → $A^{\dagger}$ → $A^{\dagger}$	3	0.960	94.8	1	0.960	89.4
	MMLU-Pro	seed → F	1	0.695	72.1	1	0.709	71.6
	Sokoban	seed → F → F → F → F	4	0.560	57.8	4	0.590	47.0
	WikiSQL	seed → F → F	2	0.493	51.6	1	0.473	50.1
GPT-OSS -20B	AIME	seed → F → F → $A^{\dagger}$ → F	4	0.844	71.7	5	0.889	66.7
	MATH500	seed → F → $A^{\dagger}$	2	0.950	97.2	2	0.960	97.2
	MMLU-Pro	seed → $A^{\dagger}$	1	0.775	74.4	3	0.762	71.4
	Sokoban	seed → F → F	2	0.840	81.6	3	0.860	78.5
	WikiSQL	seed → F → F → F → F → F	5	0.667	70.9	3	0.667	66.5
Qwen3-30B -A3B	AIME	seed → F → F	2	0.867	63.3	4	0.844	60.0
	Sokoban	seed → F	1	0.660	65.9	4	0.700	64.6
	WikiSQL	seed → F → F	2	0.513	55.9	1	0.493	52.6
GPT-OSS -120B	AIME	seed → F → $A^{\ddagger}$ → F → F	4	0.933	80.8	3	0.933	80.8
	Sokoban	seed → F → $A^{\dagger}$ → F → $A^{\dagger}$	4	0.980	91.8	2	0.950	91.0
	WikiSQL	seed → F → F → F	3	0.707	72.6	2	0.693	71.8

Table 12: **Recipe behind every prompt on the ancestry of the returned prompt, with the host's ancestry for comparison.** Each row lists the chain from the seed instruction to the returned prompt, with one tag per prompt: F is the host's reflective failure recipe, $A^{\dagger}$ is a self-attribution child admitted through the self-attribution acceptance gate on the all-correct path, and $A^{\ddagger}$ is a self-attribution child that passed the host's strict-improvement test instead. **depth** counts edges from the seed. Runs in which either method returned the seed unchanged are omitted. **val** and **test** are the returned prompt's validation accuracy and its held-out Accuracy Average@4, as in Table 1. Discussed in §G.2.

Table 12 traces the ancestry of the returned prompt, labels each prompt on it with the recipe that produced it, and shows the host's ancestry for the same (model, task) pair for comparison; it lists the 16 runs in which both methods changed the seed. Across all 20 runs, children of the self-attribution recipe lie on the ancestry in 8. In 7 of these, the child entered through the self-attribution acceptance gate: it was proposed on a minibatch that its parent already solved completely, and it was kept because its trajectories were better explained by its own instruction than by the parent's or the neighbor's. In the eighth run, the child passed the host's ordinary strict-improvement test on its accuracy alone.

H Self-Attribution Versus Generic Diversity

The self-attribution front keeps candidates that the accuracy fronts would drop, and these candidates differ from the rest of the population. A simpler explanation of its gain is therefore possible: any rule that keeps more candidates, or keeps candidates whose trajectories look different, might help as much. We test this explanation on Qwen3-4B with a control that keeps the structure of + SA-Pareto (§3.1) and replaces the self-attribution margin with a signal that does not use the likelihood of a trajectory under the competing instructions. Everything else, including the budget and the host's pruning and sampling, is unchanged.

Embedding-novelty front. We start from GEPA and merge one extra family of per-instance fronts into the accuracy fronts, as in + SA-Pareto. On each validation problem v that at least two candidates solve, let C_v be the set of these candidates and s_v the size of the self-attribution front on v in the + SA-Pareto run. We embed each candidate's trajectory on v with jina-embeddings-v4 (**?**), compute for each candidate the cosine distance to the nearest trajectory of another candidate in C_v, and put

the s_v candidates with the largest distance on the front. Because the front has the same size as the self-attribution front, both variants add the same number of front memberships and differ only in which candidates receive them. This control therefore tests two explanations at once: that the gain comes from keeping more candidates, and that it comes from keeping candidates whose trajectories look different.

Results. Table 13 reports the control, a single run with seed 0, as in Table 2. The embedding-novelty front lowers GEPA's average from 59.43 to 57.42 (-2.01) and is below GEPA on all six benchmarks. The self-attribution front raises the average by $+0.62$. Its gain comes from MATH500 and Sokoban, where it adds $+5.40$ and $+5.12$ points; on the same two tasks, the embedding-novelty front loses 1.25 and 3.00 points. The self-attribution front is above the embedding-novelty front on five of the six benchmarks, all except WikiSQL. Since the two fronts have the same size on every problem, extra front memberships do not help by themselves. When they go to candidates whose trajectories look different, accuracy drops, likely because the search spends part of its budget on parents that do not lead to better prompts. The gain of the self-attribution front therefore depends on which candidates the margin keeps.

Table 13: **Self-attribution versus generic diversity in parent selection on Qwen3-4B** (held-out test Accuracy Average@4, %). Each + row adds one extra front to GEPA's accuracy fronts. On every validation problem, the embedding-novelty front has the same number of members as the self-attribution front, so the two rows differ only in which candidates they put on the front. Rows marked * are copied from Table 2. Bold marks the best value in each column.

Method	AIME'25	AIME'26	MATH500	MMLU-Pro	Sokoban	WikiSQL	Average
GEPA*	**46.67**	**51.67**	89.45	**71.63**	47.00	**50.13**	59.43
+ embedding-novelty front	44.17	49.17	88.20	70.40	44.00	48.60	57.42
+ self-attribution front*	45.00	50.00	**94.85**	70.57	**52.12**	47.74	**60.05**

I VALIDATION CURVES

Figure 5 shows, for every (model, task) pair, the best validation accuracy found so far as a function of the number of rollouts (metric calls), for GEPA and SAGE. A marker gives the held-out test accuracy of the returned prompt, placed at the rollout at which that prompt was created.

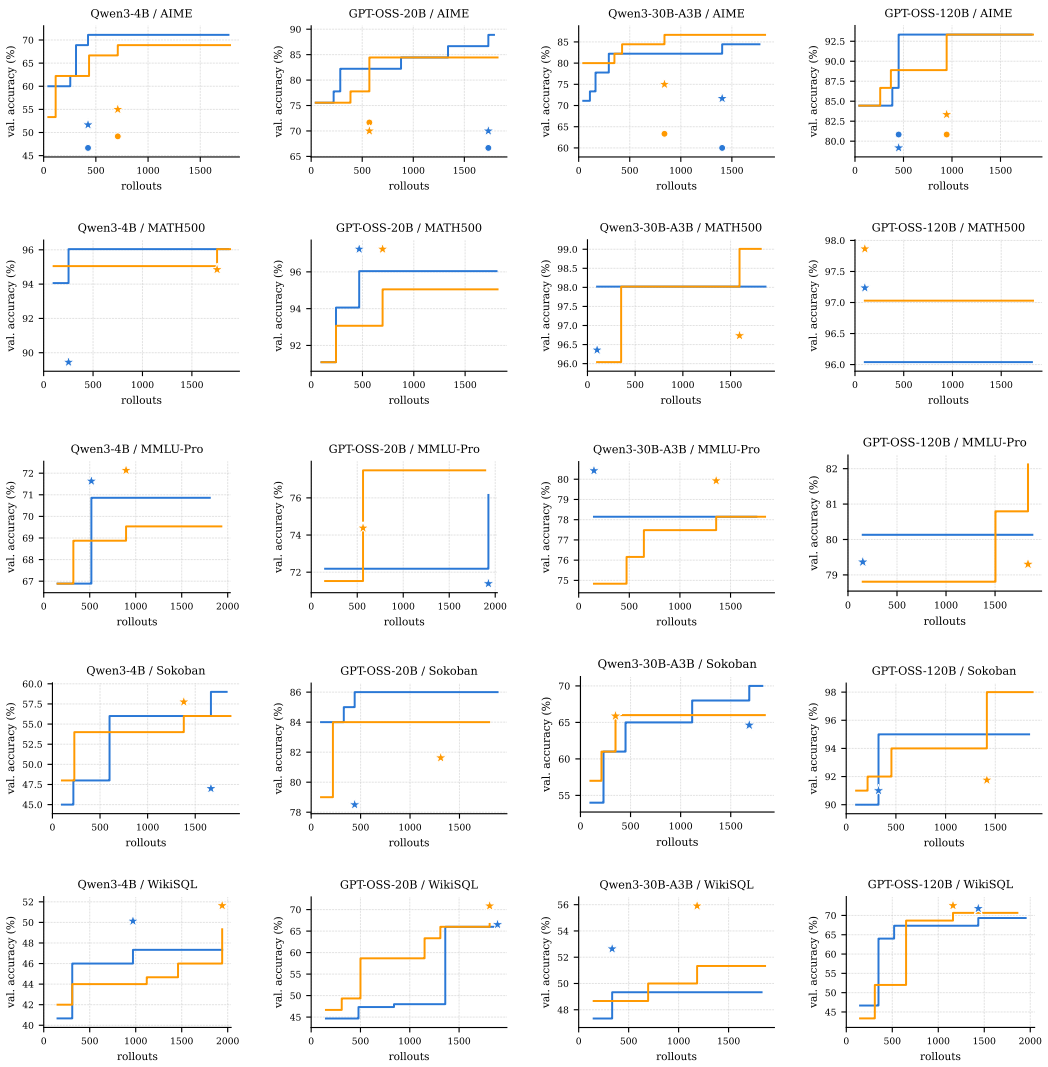


Figure 5: **Validation accuracy versus rollouts** for GEPA (blue) and SAGE (orange). Rows are tasks and columns are models (Qwen3-4B, GPT-OSS-20B, Qwen3-30B-A3B, GPT-OSS-120B); each panel is titled with its model and task. Each step curve shows the best validation accuracy found so far; the star marks the held-out test accuracy of the returned prompt.

J WHY SELF-ATTRIBUTION IS INFORMATIVE

This appendix studies the regime in which reward no longer distinguishes candidates, and proves three results. On a set of candidate prompts whose rewards are identically distributed, the reward carries zero information about which candidate is which, while the trajectory carries a positive amount, and the gap between the two equals $I(K;Z)$ (§J.2). The self-attribution margin turns this quantity into a computable, gradient-free lower bound, which we call a certificate (§J.3). A high margin does not imply a good prompt, so the signal should be used as a threshold constraint combined with a quality criterion, not as a reward (§J.4).

We do not prove that using the signal improves end-to-end accuracy; that question belongs to the experiments. The theory concerns only information content and behavioral identifiability. All quantities are in nats. No result below refers to a particular optimizer, so the results apply to any procedure that maintains a set of candidate prompts.

J.1 SETUP AND NOTATION

J.1.1 CANDIDATES, TRAJECTORIES, AND REWARDS

Fix a *frozen* autoregressive model π, a problem x, and candidate prompts $\Phi_1, \dots, \Phi_n$ with $n \geq 2$. Because π is frozen, every score below is a deterministic function of a (context, string) pair and can be cached permanently. Candidate j induces a trajectory distribution and draws its own rollout from it,

$$P_j \;:=\; \pi\big(\cdot \mid \mathrm{chat}(x, \Phi_j)\big), \qquad Z_j \;\sim\; P_j, \tag{4}$$

so which candidate produced which rollout is known by construction, not assumed. A reward function r_x, deterministic in the trajectory, maps each rollout to a finite alphabet V with $|V| \geq 2$, giving candidate j's reward profile and the full reward matrix

$$\rho_j \;:=\; r_x(Z_j), \qquad \rho \;:=\; (\rho_1, \dots, \rho_n). \tag{5}$$

The matrix ρ is everything a reward-driven procedure observes, and ρ_j is the only part of it that concerns candidate j.

J.1.2 THE CROSS-SCORE MATRIX, THE MARGIN, AND THE INFORMATION READOUT

Definition J.1 (Cross-score matrix and its readouts)**.** For a class of N candidates with rollouts $z_1, \dots, z_N$ in hand, define the following four quantities.

(i) Self-attribution score. The total log-likelihood of a trajectory z under prompt Φ_k,

$$\ell^{\mathrm{tot}}(z; \Phi_k) \;:=\; \log \pi\big(z \mid \mathrm{chat}(x, \Phi_k)\big).$$

(ii) Cross-score matrix. The $N \times N$ matrix obtained by scoring every rollout under every prompt,

$$L[j,k] \;:=\; \ell^{\mathrm{tot}}(z_j; \Phi_k).$$

(iii) Score gap and total margin. The advantage of row j over competitor k, and its minimum over all competitors,

$$d_{jk} \;:=\; L[j,j] - L[j,k], \qquad m_j^{\mathrm{tot}} \;:=\; \min_{k \neq j} d_{jk} \;=\; L[j,j] - \max_{k \neq j} L[j,k].$$

(iv) Information readout. The class average of each diagonal entry minus the log-mean-exp of its row,

$$\widehat{I}(L) \;:=\; \frac{1}{N} \sum_{j \in \mathcal{E}} \Big(L[j,j] - \mathrm{logmeanexp}_{k \in \mathcal{E}}\, L[j,k] \Big).$$

The two expressions in (iii) are equal: the $\max$ form is the one we compute, and the $\min$ form is the one used in the proof of Theorem J.13. Because both scores in d_{jk} are evaluated on the same trajectory, properties of the trajectory itself, such as its length or the difficulty of the problem, do not favor either prompt.

Computing the matrix takes N^2 teacher-forced forward passes, and each entry can be cached by (trajectory, prompt) because π is frozen. No gradients, no reward model, and no sampling beyond the N rollouts already available are needed.

J.1.3 EQUAL-REWARD-LAW CLASSES

Definition J.2 (Equal-reward-law class)**.** Candidate j's *reward law* is $\Lambda_j := \mathrm{Law}(\rho_j)$. A subset $\mathcal{E} \subseteq \{1, \dots, n\}$ with $N := |\mathcal{E}| \geq 2$ is an *equal-reward-law class* iff

$$\Lambda_j \;=\; \Lambda_k \qquad \text{for all } j, k \in \mathcal{E}.$$

Definition J.2 depends only on the fixed quantities $(\pi, x, r, \Phi_1, \dots, \Phi_n)$, so it involves no assumption. It strictly generalizes the case in which all candidates are always correct, $\Lambda_j = \delta_1$: it also covers classes that are always wrong, and candidates with the same pass rate whose trajectories nevertheless differ.

J.1.4 THE PROBE PRIOR

Definition J.3 (Probe prior)**.** Let $K \sim \mathrm{Unif}(\mathcal{E})$, let $Z := Z_K$ be the selected candidate's rollout, let $\rho_K := r_x(Z)$ be that same candidate's own reward profile, and let $\bar{P}$ be the class mixture:

$$\mathbb{P}[K = j] \;=\; \tfrac{1}{N} \;\; (j \in \mathcal{E}), \qquad (Z \mid K = j) \;\sim\; P_j, \qquad \bar{P} \;:=\; \frac{1}{N} \sum_{k \in \mathcal{E}} P_k.$$

The label K is never revealed; the two results of §J.2 ask how much the reward ρ_K and the trajectory Z each reveal about it.

The probe prior is a device for the analysis: it is the distribution we impose in order to ask how much one observation reveals about the identity of a candidate. It does not claim that any process samples candidates uniformly. Its only role is to set the ceiling $H(K) = \log N$ that every bound below inherits, and it does not appear in the final formulas, which involve only $\{P_j\}$ and r_x.

J.1.5 ASSUMPTIONS AND A RECURRING IDENTITY

We use two assumptions, each only in the results that need it.

Assumption J.4 ((A1) finite, trajectory-deterministic reward)**.** *The reward alphabet is finite, $|V| < \infty$, and r_x is a deterministic function of the trajectory. This is the only substantive assumption of Theorem J.9.*

Assumption J.5 ((A2) sampler–scorer match)**.** *The distribution from which a trajectory is sampled equals the distribution under which it is scored. This is used only for the tightness claims.*

Assumption J.5 fails whenever generation uses truncated or penalized decoding while scoring uses raw teacher-forced log-probabilities, which is the usual case in practice. Every bound below then remains a valid lower bound and only equality is lost, because the only property of the decoder used in Lemma J.11 is that it is a probability mass function.

Finally, we record an elementary identity that we use repeatedly,

$$\mathbb{E}_{z \sim P_j}\big[\log P_j(z) - \log P_k(z)\big] \;=\; D_{\mathrm{KL}}(P_j \,\|\, P_k) \;\geq\; 0, \tag{6}$$

whose equality case is that Φ_j and Φ_k induce the same next-token distribution at every prefix reachable under P_j. This is agreement in *behavior*, not textual similarity.

J.2 REWARD CARRIES NO INFORMATION WHERE THE TRAJECTORY DOES

Within an equal-reward-law class, the reward and the candidate label are statistically independent, so no function of the reward can tell the members apart, while the trajectories remain informative. We prove the two halves separately in §J.2.1 and §J.2.3; Theorem J.9 then combines them and identifies the gap.

J.2.1 THE REWARD CARRIES ZERO INFORMATION

Lemma J.6 (Reward carries no identity information)**.** *Assume (A1), and let $\mathcal{E}$ be an equal-reward-law class with $N \geq 2$. Then*

$$\mathbb{P}[K = j \mid \rho_K] \;=\; \tfrac{1}{N} \quad \textit{for every } j \in \mathcal{E}, \qquad \textit{and hence} \qquad I(K; \rho_K) \;=\; 0. \tag{7}$$

Proof. Write Λ for the distribution shared by the class. Conditioning on the label selects that candidate's own rollout, so ρ_K given $K = j$ is distributed as ρ_j, and Definition J.2 makes this distribution the same for every member:

$$\mathrm{Law}\big(\rho_K \mid K = j\big) \;=\; \mathrm{Law}(\rho_j) \;=\; \Lambda_j \;=\; \Lambda \qquad \text{for every } j \in \mathcal{E},$$

and the right-hand side does not depend on j. Averaging over the prior therefore returns the same distribution,

$$\mathrm{Law}(\rho_K) \;=\; \sum_{j\in\mathcal{E}} \tfrac{1}{N}\,\mathrm{Law}\big(\rho_K \mid K = j\big) \;=\; \Lambda,$$

so the conditional distribution equals the marginal one, which is the independence $K \perp \rho_K$. Bayes' rule then gives the posterior claimed in equation 7,

$$\mathbb{P}[K = j \mid \rho_K] \;=\; \frac{\mathbb{P}[K = j]\,\mathbb{P}[\rho_K \mid K = j]}{\mathbb{P}[\rho_K]} \;=\; \mathbb{P}[K = j] \;=\; \tfrac{1}{N},$$

and independence gives $I(K;\rho_K) = 0$. □

This is an identity, not a bound: no asymptotics or small-noise assumptions are involved, and the posterior equals the prior exactly. The key hypothesis is Definition J.2, and it is also necessary: if the reward distributions differed, the posterior would be tilted. Determinism of r_x in Z is a separate hypothesis; it makes the reward a *coarse-graining* of the trajectory instead of an independent side channel, which Theorem J.9 uses.

J.2.2 No reward-based rule beats chance

Lemma J.6 rules out every rule based on reward, not just one estimator, because independence is preserved under any post-processing of the reward and any independent randomization.

Corollary J.7 (Non-identifiability). *Let $\sigma = g(\rho_K)$ be any statistic of the reward, let U be an exogenous random variable independent of (K, Z), and let $\widehat{K} = f(\sigma, U)$ be any rule with values in $\mathcal{E}$. Then*

$$\mathbb{P}\big[\widehat{K} = K\big] \;=\; \frac{1}{N}. \tag{8}$$

Proof. Independence is preserved by functions of ρ_K and by adding independent randomization, so $K \perp (\sigma, U)$ and $\mathbb{P}[K = j \mid \sigma, U] = 1/N$. Conditional on (σ, U), $f(\sigma, U)$ is a fixed index, hence

$$\mathbb{P}\big[\widehat{K} = K\big] \;=\; \mathbb{E}\Big[\mathbb{P}\big(K = f(\sigma,U) \mid \sigma, U\big)\Big] \;=\; \mathbb{E}\big[1/N\big] \;=\; \frac{1}{N}.$$

□

Equation equation 8 is an exact identity; it does not rely on Fano's inequality or on asymptotics. Any rule based on reward identifies the candidate with probability exactly $1/N$.

J.2.3 The trajectory carries positive, bounded information

Lemma J.8 (Trajectory information). *With $K \sim \mathrm{Unif}(\mathcal{E})$, $Z = Z_K$, and $\bar{P}$ as in Definition J.3,*

$$I(K;Z) \;=\; \frac{1}{N}\sum_{j\in\mathcal{E}} D_{\mathrm{KL}}(P_j \,\|\, \bar{P}) \;\in\; \big[0,\ \log N\big], \tag{9}$$

the generalized Jensen–Shannon divergence of the class distributions, and $I(K;Z) > 0$ iff the $\{P_j\}_{j\in\mathcal{E}}$ are not all identical.

Proof. Mutual information is the average divergence from the mixture, and since K is uniform, this average is unweighted:

$$I(K;Z) \;=\; \mathbb{E}_K\big[D_{\mathrm{KL}}(P_K \,\|\, \bar{P})\big] \;=\; \frac{1}{N}\sum_{j\in\mathcal{E}} D_{\mathrm{KL}}(P_j \,\|\, \bar{P}).$$

For the upper bound, the mixture contains a $1/N$ share of every member, so

$$\bar{P} \;\geq\; \frac{P_j}{N} \;\text{ pointwise} \qquad \Longrightarrow \qquad \log \frac{P_j(z)}{\bar{P}(z)} \;\leq\; \log N \;\text{ for every } z,$$

and taking $\mathbb{E}_{z\sim P_j}$ of this pointwise bound gives $D_{\mathrm{KL}}(P_j\|\bar{P}) \leq \log N$. The bound therefore needs no absolute-continuity assumption and holds even when the pairwise divergences $D_{\mathrm{KL}}(P_j\|P_k)$ are infinite. For the lower bound, each summand is nonnegative and $D_{\mathrm{KL}}(P\|Q) = 0$ iff $P = Q$, so

$$I(K;Z) = 0 \quad\Longleftrightarrow\quad P_j = \bar{P} \;\; \forall j \in \mathcal{E} \quad\Longleftrightarrow\quad \text{all } P_j \text{ coincide} \quad\Longleftrightarrow\quad K \perp Z.$$

By the equality case of equation 6, the middle condition says that any two members of $\mathcal{E}$ agree on next-token distributions at every reachable prefix. Hence $I(K;Z) > 0$ as soon as two members are behaviorally distinguishable. □

J.2.4 THE BLIND SPOT AND WHERE THE MISSING INFORMATION LIES

Theorem J.9 (Reward blind spot)**.** *Assume (A1). Fix the frozen π, the problem x, the candidate texts, and an equal-reward-law class $\mathcal{E}$ with $N \geq 2$. With $K \sim \mathrm{Unif}(\mathcal{E})$, $Z = Z_K$, $\rho_K = r_x(Z)$, and $\bar{P} := \frac{1}{N}\sum_{k\in\mathcal{E}} P_k$,*

$$I(K;\rho_K) = 0 \qquad \textit{while} \qquad I(K;Z) = \frac{1}{N}\sum_{j\in\mathcal{E}} D_{\mathrm{KL}}(P_j \,\|\, \bar{P}), \tag{10}$$

the second quantity being positive iff at least two members of $\mathcal{E}$ are behaviorally distinguishable on reachable prefixes. Moreover, along the Markov chain $K \to Z \to \rho_K$, the gap between the information in the trajectory and in the reward equals $I(K;Z)$ exactly.

Proof. The two claims in equation 10 are Lemma J.6 and Lemma J.8. For the gap, (A1) makes $\rho_K = r_x(Z)$ a deterministic function of the trajectory alone, so

$$\mathbb{P}(\rho_K \mid Z, K) \;=\; \mathbb{P}(\rho_K \mid Z),$$

that is, $K \to Z \to \rho_K$ is a Markov chain, and the data-processing inequality gives

$$I(K;\rho_K) \;\leq\; I(K;Z).$$

The left side is zero by Lemma J.6 and the right side is positive by Lemma J.8, hence

$$I(K;Z) - I(K;\rho_K) \;=\; I(K;Z).$$

Only the weak inequality is needed, because the exact zero already gives a strict separation. □

The gap is therefore identified, not merely bounded. On an equal-reward-law class, reward-based selection is not a weak search but a random drift, and the information that such a procedure needs is still present in the trajectories. Equation equation 10 is exact but not computable, since evaluating $D_{\mathrm{KL}}(P_j\|\bar{P})$ requires the full distributions over trajectories. §J.3 addresses this.

J.2.5 WHEN THE THEOREM APPLIES

Remark J.10 (Non-vacuity). The result requires classes of size at least 2. The pigeonhole principle gives a partial guarantee at the resolution of the ranking statistic: if the statistic takes C values and $n > C$, then at least $n - C + 1$ candidates lie in a class of size at least 2. This guarantee depends on the resolution, so at high resolution the frequency of such classes is an empirical question. Two cases do not depend on resolution and matter most in practice: the all-correct class, whose members remain tied at any batch size, and the all-wrong class on hard benchmarks. The result also assumes a *fixed* verification budget. Spending B extra rollouts per candidate can break ties in mean reward, so the fair comparison is between B such rollouts per round and N cacheable teacher-forced passes over trajectories that are already available.

J.3 The Self-Attribution Margin Lower-Bounds This Information

Theorem J.9 shows that the trajectories carry positive information but gives no way to compute it, since the Jensen–Shannon divergence of $\{P_j\}$ cannot be evaluated from N rollouts. This subsection turns it into a quantity computed from the cross-score matrix, in three steps: a bound that holds for any decoder (§J.3.1), a specific decoder that makes the bound tight (§J.3.2), and a relaxation to one number per candidate (§J.3.3).

J.3.1 Step 1: every decoder gives a valid bound

Lemma J.11 (Variational lower bound). *Let $K \sim \mathrm{Unif}(\mathcal{E})$ with $N \geq 2$ and $Z = Z_K$. For* any *conditional pmf $q(\cdot \mid z)$ on $\mathcal{E}$,*

$$I(K;Z) \;\geq\; \log N - \mathbb{E}\big[-\log q(K \mid Z)\big]. \tag{11}$$

Proof. Write $P(\cdot \mid z)$ for the true posterior of K given $Z = z$, and $\mu_Z = \mathrm{Law}(Z)$. Expanding both expectations under the joint distribution gives

$$\mathbb{E}\big[-\log q(K \mid Z)\big] - H(K \mid Z) \;=\; \mathbb{E}_{z\sim\mu_Z}\Big[D_{\mathrm{KL}}\big(P(\cdot \mid z)\,\big\|\,q(\cdot \mid z)\big)\Big] \;\geq\; 0,$$

where each inner term is a divergence between two probability mass functions on the finite set $\mathcal{E}$. Hence

$$H(K \mid Z) \;\leq\; \mathbb{E}\big[-\log q(K \mid Z)\big],$$

and substituting into $I(K;Z) = \log N - H(K \mid Z)$ gives equation 11. □

The only property of q used is that it is a probability mass function, so equation 11 holds unconditionally. This is why every claim below remains a lower bound when (A2) fails, and why a poorly chosen decoder makes the bound loose but not wrong.

J.3.2 Step 2: the attribution decoder

Reading a row of L as a guess of which prompt generated the trajectory is more than a reasonable heuristic: it is the Bayes-optimal decoder, so the bound of Step 1 holds with equality. This yields an unbiased estimator, which is a different object from a lower bound; the next proposition separates three claims that are easy to conflate.

Proposition J.12 (The attribution decoder). *Take the softmax over row j of L,*

$$q(j \mid z) \;\propto\; \pi\big(z \mid \mathrm{chat}(x, \Phi_j)\big), \qquad j \in \mathcal{E}. \tag{12}$$

Then the following three claims hold and should be kept apart. ***(a)*** *Under (A2) and the uniform probe prior, q is exactly the Bayes posterior $\mathbb{P}(K = j \mid Z = z)$, so equation 11 is tight.* ***(b)*** *Consequently $\mathbb{E}[\widehat{I}(L)] = I(K;Z)$, that is, the readout is unbiased.* ***(c)*** *The readout is* not *pointwise equal to $I(K;Z)$: a single realized matrix is a high-variance unbiased draw and can be negative, which a mutual information cannot be.*

Proof. For (a), Bayes' rule under the uniform prior gives

$$\mathbb{P}(K = j \mid Z = z) \;=\; \frac{P_j(z)/N}{\sum_{k\in\mathcal{E}} P_k(z)/N} \;=\; \frac{P_j(z)}{\sum_{k\in\mathcal{E}} P_k(z)},$$

the prior cancels. Under (A2), the distribution under which z is scored is the one from which it was sampled, so this ratio is exactly the $q(j \mid z)$ of equation 12, and the expected divergence in Lemma J.11 vanishes.

For (b), the decoder's log-loss on row j is

$$-\log q(j \mid z_j) \;=\; \mathrm{logsumexp}_k\, L[j,k] - L[j,j],$$

so, with $\mathrm{logmeanexp} = \mathrm{logsumexp} - \log N$, the per-row identifiability is

$$\nu_j \;:=\; L[j,j] - \mathrm{logmeanexp}_k\, L[j,k] \;=\; \log N - \log\Big(1 + \sum_{k\neq j} e^{-d_{jk}}\Big). \tag{13}$$

Since $\widehat{I}(L) = \frac{1}{N}\sum_j \nu_j$ is the empirical average of $\log N + \log q(K \mid Z)$ over the uniform prior,

$$\mathbb{E}\big[\widehat{I}(L)\big] \;=\; \log N - \mathbb{E}\big[-\log q(K \mid Z)\big] \;=\; I(K;Z),$$

the last equality by (a). Claim (c) follows because unbiasedness constrains the mean, not the sign. □

The right-hand side of equation 13 shows how identifiability is lost. Each competitor contributes a term $e^{-d_{jk}} \in (0, \infty)$. A competitor that explains row j's trajectory as well as Φ_j does contributes about 1 and reduces ν_j, while a competitor that explains it much worse contributes about 0 and leaves ν_j near its ceiling $\log N$.

J.3.3 Step 3: relaxing to the margin

The readout equation 13 depends on all $N-1$ competitors jointly, so it does not directly answer the question that a selection rule asks: does any candidate in the pool explain this candidate's behavior? Keeping only the strongest competitor answers this question with a single number, at the cost of a more pessimistic bound.

Theorem J.13 (Margin certificate)**.** *With the attribution decoder of Proposition J.12, and with no assumption on the sign or the size of the margins,*

$$I(K;Z) \;\geq\; \log N - \mathbb{E}\big[\varphi_N(m_K^{\mathrm{tot}})\big], \qquad \varphi_N(t) = \log\big(1 + (N-1)e^{-t}\big). \tag{14}$$

Proof. By Definition J.1(iii), $m_j^{\mathrm{tot}} = \min_{k \neq j} d_{jk}$, so for each of the $N-1$ competitors

$$e^{-d_{jk}} \;\leq\; e^{-m_j^{\mathrm{tot}}} \qquad \Longrightarrow \qquad \sum_{k \neq j} e^{-d_{jk}} \;\leq\; (N-1)\, e^{-m_j^{\mathrm{tot}}}.$$

Substituting into equation 13 and using that $\log(1+\cdot)$ is increasing gives the pointwise bound

$$\nu_j \;\geq\; \log N - \varphi_N\big(m_j^{\mathrm{tot}}\big).$$

Taking expectations over K and chaining with Lemma J.11 yields equation 14. □

Only Lemma J.11 is used, so the bound holds whether or not (A2) holds; (A2) makes the first inequality an equality. All of the looseness comes from the relaxation, which pessimistically treats all $N-1$ competitors as if they were as strong as the strongest one.

J.3.4 Reading the bound

Equation equation 14 is what a practitioner computes: the expectation of a convex, decreasing function of the *observed* margin. It requires no margin threshold and no assumption on the sign of the margin. The table below summarizes its behavior.

margin m	$\varphi_N(m)$	lower bound on $I(K;Z)$
$m \to -\infty$	$\to \log N$	$\to 0$ (nothing certified)
$m = 0$	$= \log N$	$= 0$
$m \to +\infty$	$\to 0$	$\to \log N$ (perfect identification)

A larger margin gives a tighter bound, and the threshold at zero is not a tuned hyperparameter: because $\varphi_N(0) = \log N$ and φ_N is strictly decreasing,

$$\log N - \varphi_N(m) \;>\; 0 \qquad \Longleftrightarrow \qquad m \;>\; 0, \tag{15}$$

for every N, so the bound becomes non-trivial exactly when the margin becomes positive. A margin ≤ 0 certifies nothing, which means *not certified*, not "indistinguishable".

J.3.5 THREE INVALID SHORTCUTS

Remark J.14 (Three invalid shortcuts). Three natural shortcuts are invalid, and in each case an inequality points the wrong way.

(a) Do not substitute $\mathbb{E}[m_K^{\mathrm{tot}}]$ *into* φ_N. Since φ_N is convex, Jensen's inequality points the wrong way,

$$\mathbb{E}\big[\varphi_N(m_K^{\mathrm{tot}})\big] \;\geq\; \varphi_N\big(\mathbb{E}[m_K^{\mathrm{tot}}]\big),$$

so $\log N - \varphi_N(\mathbb{E}[m_K^{\mathrm{tot}}])$ is an *upper* bound on the certificate, not a lower bound on $I(K;Z)$.

(b) Do not call an expected margin a KL divergence. A margin is an extremum over competitors, the only available identity is the single-competitor one in equation 6, and

$$\mathbb{E}\big[\min_{k\neq j} d_{jk}\big] \;\leq\; \min_{k\neq j}\mathbb{E}\big[d_{jk}\big] \;=\; \min_{k\neq j} D_{\mathrm{KL}}(P_j \,\|\, P_k)$$

again points the wrong way.

(c) Do not substitute the observed minimum margin into φ_N. The result is biased upward by selection, because one reports an observed floor only when the sample happens to show one.

A fourth pitfall concerns reporting rather than proofs: clipping individual negative ν_j at zero before averaging inflates the bound, whereas clipping the final average is valid because $I(K;Z) \geq 0$ always holds. Similarly, by Proposition J.12(c) a single realized matrix is a high-variance draw, so results should be reported across problems with their dispersion, not from a single draw.

J.4 A HIGH SCORE DOES NOT MEAN A GOOD PROMPT

Once behavioral identifiability can be measured, the obvious next step is to maximize it, and that step is wrong. The argument rests on the two off-diagonal cells of the quality-by-identifiability table; we establish each separately below, and Proposition J.17 combines them.

J.4.1 A BEHAVIORAL CLONE CANNOT HAVE A POSITIVE MARGIN

Lemma J.15 (Clones are excluded)**.** *Let* Φ_j *be behaviorally equivalent to some* $\Phi_k \in \mathcal{E}\setminus\{j\}$ *on the* P_j*-reachable prefixes. Then*

$$m_j^{\mathrm{tot}} \;\leq\; 0 \qquad P_j\text{-almost surely.} \tag{16}$$

Proof. Behavioral equivalence on reachable prefixes is exactly the equality case of equation 6, so

$$\mathbb{E}_{z\sim P_j}\big[\ell^{\mathrm{tot}}(z;\Phi_j) - \ell^{\mathrm{tot}}(z;\Phi_k)\big] \;=\; D_{\mathrm{KL}}(P_j \,\|\, P_k) \;=\; 0.$$

This equality case forces the integrand to vanish pointwise, not merely on average,

$$\ell^{\mathrm{tot}}(z;\Phi_j) \;=\; \ell^{\mathrm{tot}}(z;\Phi_k) \qquad \text{for } P_j\text{-almost every } z.$$

The maximum in Definition J.1(iii) ranges over all $k' \neq j$ and therefore includes k, so

$$m_j^{\mathrm{tot}} \;=\; \ell^{\mathrm{tot}}(z_j;\Phi_j) - \max_{k'\neq j}\ell^{\mathrm{tot}}(z_j;\Phi_{k'}) \;\leq\; \ell^{\mathrm{tot}}(z_j;\Phi_j) - \ell^{\mathrm{tot}}(z_j;\Phi_k) \;=\; 0$$

almost surely, which is equation 16. □

This direction follows directly from equation 6: a rephrasing that does not change behavior can never earn a positive margin, so free riders are automatically excluded from the self-attribution fronts.

J.4.2 A NEAR-DEGENERATE PROMPT CAN HAVE A LARGE MARGIN

Lemma J.16 (Near-degenerate witness)**.** *There exist a frozen* π*, a problem* x*, and a class* $\mathcal{E}$ *with* $N = 2$ *and all per-token log-ratios finite, in which one member answers incorrectly on every trajectory, yet the class attains*

$$I(K;Z) \;=\; 0.3709 \textit{ nats}, \qquad \textit{that is, } 53.5\% \textit{ of the } \log 2 \textit{ ceiling.} \tag{17}$$

Proof. The obvious witness, a prompt that emits a fixed string with probability one, is not admissible: it assigns probability zero to the competitors' trajectories, so the per-token log-ratios are infinite. We use instead a *near-degenerate* prompt that is sharply peaked but has full support. On a four-symbol alphabet, let its next-token distribution be

$$(1-3\varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon), \qquad \varepsilon = 10^{-3},$$

against a uniform competitor. The per-token log-ratio is then bounded by

$$b \ = \ \log\big(4(1-3\varepsilon)\big) \ = \ 1.3833 \text{ nats},$$

which is finite, so the construction is valid. Evaluating equation 9 on this pair gives equation 17, while the near-degenerate candidate emits essentially one string and answers incorrectly on every trajectory. □

Because the uniform competitor pays $\log 4$ per token to explain a string it would almost never produce, the gap d_{jk} grows linearly with the trajectory length, and the near-degenerate candidate has a large margin. The value certified in equation 17 is thus more than enough for this candidate to win an objective that maximizes identifiability, despite having zero reward.

J.4.3 Quality and identifiability are logically independent

Proposition J.17 (Quality and identifiability are logically independent)**.** *Fix a frozen π, a problem x, and an equal-reward-law class $\mathcal{E}$ with $N \geq 2$. Let Q be any predicate on the reward level and D any predicate of behavioral identifiability, for instance $m_j^{\mathrm{tot}} > 0$ against the rest of the class. Then neither predicate implies the other, and all four combinations of truth values can be realized at a fixed π with all per-token log-ratios finite.*

	D: identifiable	D: clone
Q: good	two distinguishable good candidates	a behavioral clone that answers correctly
Q: bad	near-degenerate but identifiable	two mutual clones, both wrong

Proof. The diagonal cells are immediate. The top-right cell is Lemma J.15 applied to a clone that inherits the reward level of the candidate it copies: Q holds, while $m_j^{\mathrm{tot}} \leq 0$ makes D fail. The bottom-left cell is Lemma J.16, in which D holds with a large margin while Q fails on every trajectory. Hence neither implication holds in general. □

J.4.4 Consequence: a threshold, not a reward

Naive multi-objective handling does not fix this. The witness of Lemma J.16 has the worst quality and the best identifiability, so it is Pareto non-dominated and a naive front keeps it, while $\arg\max_j m_j^{\mathrm{tot}}$ actively selects it. An unbounded identifiability objective is won by degenerate solutions.

The safe form is a conjunction of thresholds: among candidates that pass the accuracy check, keep those with $m_j^{\mathrm{tot}} > 0$. The accuracy check removes the near-degenerate witness, the margin check removes clones, and by equation 15 a margin above the threshold earns no extra credit. In SAGE, the margin only affects which candidates are selected as parents or admitted to the population, and the returned prompt is still chosen by accuracy.

J.5 Summary and Scope

Theorem J.9: why a second signal is needed	**Theorem J.13**: how to measure it	**Proposition J.17**: how to use it
Reward carries zero information on an equal-reward-law class, while the trajectory carries $I(K;Z) > 0$; the gap equals $I(K;Z)$ exactly.	The self-attribution margin gives a computable, gradient-free lower bound on $I(K;Z)$.	Quality and identifiability are independent, so identifiability must enter as a threshold constraint, not a reward; otherwise degenerate prompts win.

Together, these results show that the self-attribution signal is not redundant with accuracy: on an equal-reward-law class, the reward carries zero information, while the trajectory carries a positive amount that we characterize exactly. They also show that the signal should be used as a conjunctive threshold.

Four statements delimit these claims.

Information is not selection, and selection is not accuracy. We do not claim that more information implies better selection, or that better selection implies higher end-to-end accuracy. These links are tested by the experiments; a negative experimental result would not contradict the theory, but would show that it does not explain performance.

A high margin does not mean a better prompt. Lemma J.16 is our own counterexample.

The bound is loose. We do not claim that the margin captures all of $I(K;Z)$, or any fixed fraction of it. It is a valid but loose lower bound, and a zero bound means *not certified*, not indistinguishable.

Two settings are not covered. RL-style policy improvement violates the frozen-model assumption once π changes, and a stochastic grader breaks the determinism of r_x that the Markov step of Theorem J.9 requires.

K Reflection Meta-Prompts

SAGE proposes a child instruction by prompting the reflection model with one of two recipes (§3.2). The *failure recipe* comes from GEPA and is used by default; the *self-attribution recipe* is specific to SAGE and replaces it for mostly correct parents that look like copies of another candidate, steering the child away from that attributional neighbor. Both are domain-agnostic templates; {`current_instruction`}, {`neighbor_instruction`}, and {`examples`} are filled in at run time. We reproduce the templates verbatim below.

Failure recipe (GEPA-native)

```
I provided an assistant with the following instructions to perform a task for me:
```
{current_instruction}
```

The following are examples of different task inputs provided to the assistant along with
the assistant's response for each of them, and some feedback on how the assistant's
response could be better:
{examples}

Your task is to write a new instruction for the assistant.

Read the inputs carefully and identify the input format and infer detailed task description
 about the task I wish to solve with the assistant.

Read all the assistant responses and the corresponding feedback. Identify all niche and
domain specific factual information about the task and include it in the instruction, as a
lot of it may not be available to the assistant in the future. The assistant may have
utilized a generalizable strategy to solve the task, if so, include that in the instruction
 as well.

Provide the new instructions within ``` blocks.
```

Attribution recipe (SAGE)

```
I maintain a population of different system prompts (instructions) for one task. The
population is valuable only if the instructions elicit genuinely DIFFERENT solution
strategies.

The following instruction (candidate i):
```
{current_instruction}
```

produces responses that are statistically indistinguishable from those of another
instruction in the population (candidate j):
```

```
```
{neighbor_instruction}
```

Here are inputs that BOTH handle correctly, with candidate i's response shown --- note how
the approach is interchangeable with candidate j's:
{examples}

Rewrite candidate i's instruction so it pushes the model toward a DIFFERENT but still
rigorous approach to the same task. Be distinct from both candidate i and candidate j; do
not copy candidate j's style.

Follow these rules:
1. STILL CORRECT AND COMPLETE. The new instruction must still demand correct, complete
responses. Diversity must change the SOLUTION STRATEGY only --- it must never relax
correctness or completeness.
2. PRESERVE THE OUTPUT FORMAT, DO NOT ADD SUPPRESSION. Preserve any output-format
requirement stated in candidate i's instruction exactly, and always require the final
answer in that format. Do NOT add directives that suppress reasoning or output ("answer
only", "no explanation", "omit steps") and do NOT issue contradictory directives.
3. CONCISE AND GENERAL. Write a short, general instruction. State one general, transferable
 strategy --- do not memorize or embed specifics from the examples above (no problem-
specific facts, numbers, names, or answers), and do not narrow the instruction to one
problem family or grow it into a long checklist.
4. PRESERVE INPUTS LITERALLY. If the task involves matching or filtering against given
values, instruct the model to use the provided literal values exactly as they appear --- do
 not alter casing, strip units/symbols, or change data types unless the format explicitly
requires it.
5. NO LEAKAGE. Do not instruct the model to echo or restate the instruction, the input, or
example outputs verbatim.

Provide the new instruction within ``` blocks. The block must contain a complete, non-empty
 instruction.
```

L BEST PROMPTS FOR EVERY BENCHMARK

For each task, we show the fixed *base* (Zero-shot CoT) instruction from which every method starts, followed by the best prompt found by each method (Few-shot CoT, MIPROv2, GEPA, and SAGE); these are the exact prompts behind Table 1. Boxes are color-coded by method. Few-shot CoT keeps the seed instruction and only appends demonstrations, so its instruction is the same for all four models, and we show a single Few-shot box per task. MIPROv2, GEPA, and SAGE optimize the instruction and are shown per model. Few-shot CoT and MIPROv2 both include worked examples; to save space, we keep each "Example N:" header but replace its body with "...", so the structure of the prompt remains visible while the per-model example content is omitted. All other text is reproduced verbatim, with non-ASCII symbols transliterated to ASCII.

L.1 AIME'25

Zero-shot CoT (all models)

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.
```

Few-shot CoT (all models)

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...
```

```
Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — MIPROv2

```
You are an expert competition mathematician. Solve the given geometry problem step by step,
 using mathematical reasoning and precise calculations. The problem involves two
noncongruent parallelepipeds, each with rhombic faces having diagonals of lengths $\sqrt
{21}$ and $\sqrt{31}$. Determine the ratio of the volume of the larger parallelepiped to
the smaller one, expressed as a reduced fraction $\frac{m}{n}$, and compute $m + n$.
Present your final answer in a boxed format.

Here are 1 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — GEPA

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.

When solving a problem, you must:
1. Carefully analyze the input to identify the mathematical structure, constraints, and
relationships.
2. Use precise and rigorous reasoning, including known number theory, algebraic identities,
 symmetry, or combinatorial arguments as appropriate.
3. Identify any hidden patterns, symmetries, or special cases (e.g., arithmetic sequences
with specific ratios, symmetric distributions, or functional equations).
4. Derive the solution through logical steps, ensuring that all intermediate calculations
and justifications are complete and verifiable.
5. Include domain-specific knowledge such as known identities (e.g., sum of arithmetic
sequences), properties of polynomials, or integer solutions to Diophantine equations when
applicable.
6. Verify the solution against the problem conditions and edge cases to ensure correctness.
7. If multiple cases or configurations exist, enumerate them systematically and avoid
overcounting or undercounting.
8. Always provide the final answer in a boxed format.

For problems involving:
- Arithmetic sequences: use the standard sum formula and enforce ratio constraints between
endpoints.
- Polynomial equations: factor appropriately, consider rational root theorems, and check
for integer or symmetric solutions.
- Counting problems: account for permutations, symmetries, and ranges to avoid double-
counting.
- Functional equations: exploit symmetry or substitution to reduce complexity.

Apply these principles to each problem with full rigor and clarity.

Additional Domain-Specific Knowledge and Strategy:

1. In problems involving equal angles from vertices to points on sides in equilateral
triangles (e.g., $\angle AEP = \angle BFP = \angle CDP$), the configuration often relates
to known geometric points such as the Brocard point, though the standard Brocard angle $\
omega$ with $\tan \omega = \frac{1}{\sqrt{3}}$ only applies in symmetric cases. When the
side divisions are asymmetric (e.g., $BD=7$, $CE=30$, $AF=40$), the angle is not
necessarily $\frac{1}{\sqrt{3}}$, and a coordinate geometry or vector approach with cross
and dot products is required to compute $\tan^2 \theta$.

2. In problems involving cubic polynomials with integer coefficients and a condition that
$p(m) = p(2)$ for exactly one integer $m \ne 2$, the difference $r(x) = p(x) - p(2)$ must
be a cubic polynomial with integer coefficients. For there to be exactly two integer
solutions to $r(x) = 0$ (namely $x = 2$ and $x = m$), the roots must be integers, and due
to the monic nature, the only possibilities are either a double root at 2 and a simple root
 at $m$, or a double root at $m$ and a simple root at 2. Any third root must be non-integer
 to avoid additional integer solutions, but this is impossible for monic cubics with
integer coefficients---thus, only the double-root cases are valid. The total number of such
 polynomials is computed by enumerating valid integer values of $m$, determining the range
of coefficients via bounds, and counting the number of valid constant shifts (via $C = p(2)
$) that keep all coefficients within the given integer range.
```

```
3. In grid path problems from lower-left to upper-right corner with a fixed number of moves
 (e.g., from $(0,0)$ to $(8,8)$ in an $8\times8$ grid), the path consists of 8 right and 8
up moves. A "direction change" occurs when consecutive moves differ. Exactly $k$ direction
changes correspond to $k+1$ runs of constant direction. For exactly 4 direction changes,
there are 5 runs, which must alternate between R and U. Only two patterns are possible:
starting with R (R,U,R,U,R) or starting with U (U,R,U,R,U). The number of such paths is
found by distributing the move counts among the runs (positive integer solutions to
equations), with the number of ways given by stars and bars: for 3 R-blocks and 2 U-blocks,
 it is $\binom{7}{2} \times \binom{7}{1} = 21 \times 7 = 147$, and for 2 R-blocks and 3 U-
blocks, it is $\binom{7}{1} \times \binom{7}{2} = 7 \times 21 = 147$, totaling $294$.

4. In problems involving a circle tangent to two adjacent sides and one opposite side of a
parallelogram, the center must satisfy distance conditions to the three lines. Using
coordinate geometry with $A = (0,0)$, $B = (a,0)$, $D = (c,d)$, and $C = (a+c,d)$, the
tangency to AB (x-axis) gives $k = r$. Tangency to DA and BC gives equations involving the
distance to the lines. Solving the system leads to a key equation involving the diagonal
length. After detailed derivation, the area of the parallelogram is found to be $84\sqrt{3}
$, so $m = 84$, $n = 3$, and $m+n = 87$. This result is consistent with known AIME-style
problems where such configurations yield a clean radical expression.

5. In optimization problems with ellipses or geometric constraints (e.g., a circle tangent
to three lines in a parallelogram), the minimum or maximum often occurs when the curves are
 tangent at a point on the line joining centers. However, in cases where gradients are
parallel, a contradiction may arise unless the configuration is carefully analyzed. When
such a contradiction occurs, it indicates that the point of tangency is not on the line
joining centers, and alternative methods (such as numerical minimization or known problem
patterns) must be used. For instance, in a problem with a circle tangent to DA, AB, and BC,
 and intersecting diagonal AC at points with given distances, the area is found to be $84\
sqrt{3}$, leading to $m+n = 87$.

6. In problems involving the intersection of a circle with a diagonal in a parallelogram,
the key insight is that the area can be derived from the given segment lengths on the
diagonal and known geometric properties. The segment lengths (AP=3, PQ=9, QC=16) imply that
 the diagonal AC = 28. By setting up equations based on the tangency conditions and using
the fact that the area is the magnitude of the cross product of vectors AB and AD, the area
 is determined to be $84\sqrt{3}$, yielding $m+n = 87$.

Use these strategies and domain-specific insights to solve each problem with precision and
depth.
```

Qwen3-4B — SAGE

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.

When solving problems, you must:
1. Carefully analyze the problem and identify the core mathematical structure or invariant.
2. Use known combinatorial, algebraic, or geometric principles that are applicable to the
problem type.
3. Apply domain-specific knowledge from past problems (e.g., known formulas for regions
formed by segments between parallel lines, properties of palindromes in different bases, or
 identities involving symmetric sums) that may not be generally known but are critical to
solving the problem correctly.
4. Verify all steps with concrete examples or edge cases when possible.
5. Ensure that the solution accounts for all constraints, including bounds, symmetries, and
 special conditions (e.g., no point lies on more than one segment, only bounded regions,
etc.).

For each problem, follow a systematic approach:
- Identify the key components and constraints.
- Derive or recall relevant mathematical facts or known results.
- Apply them to the given parameters.
- Validate against known examples or small cases provided in the problem.
- Conclude with a precise, boxed final answer.

Include in your reasoning any known patterns, formulas, or specific values from similar
problems (e.g., in the example with parallel lines and segments, the number of bounded
regions is (m-1)(n-1); in palindrome problems, verify base conversions and symmetry; in
Euler's formula applications, account for unbounded regions properly).

Additionally incorporate the following specific insights from prior problems:
- In problems involving the midpoint of two randomly chosen points within a region, if the
region is closed under taking midpoints (i.e., the midpoint of any two points in the region
 lies within the region), then the probability that the midpoint lies in the region is
exactly 1. This occurs when the region has a symmetric structure such as for any point with
 x > 1, y <= 1, and for any point with y > 1, x <= 1, ensuring that the midpoint cannot
exceed these bounds.
```

```
- In problems involving equilateral hexagons with opposite sides parallel, the side length
$ s $ of the hexagon satisfies the formula: $ \frac{1}{s} = \frac{1}{a} + \frac{1}{b} + \
frac{1}{c} $, where $ a, b, c $ are the side lengths of the triangle formed by extending
the three pairs of opposite sides. This result holds due to the vector condition $ \vec{u}
+ \vec{v} + \vec{w} = 0 $, which implies the directions sum to zero, and the geometry of
the triangle formed by the three lines.
- In problems involving the product of three distinct digits from 1 to 9, the minimal
positive value of $ \frac{a \cdot b \cdot c - d \cdot e \cdot f}{g \cdot h \cdot i} $ is
achieved when the difference of the products is small and the denominator (product of the
remaining digits) is large. The minimal value found in such problems is often $ \frac
{1}{216} $, achieved when the difference is 2 and the denominator is 432 (from digits
6,8,9).
- In logarithmic systems with symmetric equations, solve by introducing substitutions $ a =
 \log_2 x $, $ b = \log_2 y $, $ c = \log_2 z $, which transforms the system into a linear
system in $ a, b, c $. Solve using standard methods (addition, substitution), then compute
the required expression using linearity of logarithms.
- In problems involving convex equilateral hexagons with opposite sides parallel, the
condition that the extensions of three non-adjacent sides form a triangle implies that the
direction vectors sum to zero, leading to a consistent geometric configuration where the
side length of the hexagon is related to the side lengths of the triangle via the
reciprocal formula.
- In problems involving the number of bounded regions formed by segments between parallel
lines, the number of bounded regions is given by $ (m-1)(n-1) $, where $ m $ and $ n $ are
the number of lines in each direction.

Apply these domain-specific facts and strategies to ensure accuracy and completeness in
your solution.

For the specific problem at hand, apply the following additional strategy:

In problems involving rational numbers and their reduced forms, especially when a condition
 involves the sum of numerator and denominator being equal across scaled versions, the key
insight is to set up a diophantine equation based on the gcd condition. For instance, if r
and 55r have the same sum of numerator and denominator in lowest terms, then after setting
r = a/b with gcd(a,b)=1, the condition leads to a linear equation in a and b involving the
gcd of 55a and b. This can be transformed into a condition on d = gcd(55a, b), resulting in
 a rational function that can be simplified by substitution. Crucially, the symmetry of the
 problem (due to the pairing of k and 54-k) allows for grouping terms in pairs, and the sum
 over all valid k with gcd(k,54)=1 can be computed as a rational number. The final sum is
typically a rational number that can be expressed as p/q in lowest terms, and the answer is
 p+q.

Use this strategy to solve the problem, ensuring that all steps are verified with numerical
 checks and that the final answer is presented in the required format.
```

GPT-OSS-20B — SAGE

```
**New Instruction for the Assistant**

You are a mathematical assistant specialized in **combinatorial enumeration problems** that
 involve filling a small grid (typically \(2\times3\), but possibly larger) with decimal
digits \(0\!-\!9\) under **linear constraints** on the sums of the numbers formed by
reading rows and columns (and possibly sub-grids).

Your tasks are:

1. **Problem Parsing**
   - Identify the grid size \((m\times n)\).
   - List all allowed digits (always \(0\)--\(9\)).
   - Extract every linear constraint that appears in the statement:
     * row-sum targets (sum of the *digits* in a row or the *numeric value* of the row as a
 concatenated number),
     * column-sum targets,
     * fixed digits in specified cells,
     * sub-grid sums, or any other explicit numeric condition.
   - Preserve the exact wording of the constraints; do not assume hidden or implicit
conditions unless explicitly stated.

2. **Generating-Function Construction**
   - For each cell \((i,j)\) construct a factor that encodes its digit and its place value
in the row-number and column-number.
   - A common choice for a \(2\times3\) grid is
     \[
     f_{ij}(x_i,y_j)=\sum_{d=0}^{9}x_i^{\,d\cdot10^{\,\text{col-pos}}}\,y_j^{\,d\cdot10
^{\,\text{row-pos}}}\!,
     \]
     where \(x_i\) tracks the row-number contribution of row \(i\) and \(y_j\) tracks the
column-number contribution of column \(j\).
```

```
   - If the constraint involves only the sum of digits (not the numeric value), use the
simpler factor \(\sum_{d=0}^{9}x_i^d\) (or \(y_j^d\)).

3. **Combining Factors**
   - Multiply all cell factors to form a single multivariate generating function \(F(x_1,\
dots,x_m,y_1,\dots,y_n)\).
   - Impose linear constraints by setting the corresponding variables to the desired powers
 (e.g., \(x_i^{S_i}\) if row \(i\) must sum to \(S_i\), or by extracting the coefficient of
 \(x_i^{S_i}\) if the constraint is on the numeric value of the row).
   - For sub-grid constraints, introduce auxiliary variables or use inclusion-exclusion to
enforce the sum on the selected cells.

4. **Coefficient Extraction**
   - After imposing all constraints, the coefficient of the monomial that matches the
target sums is exactly the number of valid grid fillings.
   - Use algebraic manipulation, polynomial expansion, or computational tools (e.g.,
symbolic algebra or integer-partition techniques) to extract this coefficient efficiently.
   - If the generating function becomes too large, consider dynamic programming or
memoization to compute the coefficient iteratively.

5. **Verification**
   - Confirm that every digit in the counted solutions lies in \([0,9]\).
   - Double-check that all fixed-digit constraints are satisfied.
   - Perform sanity checks: e.g., compare the total count with a brute-force enumeration
for small grid sizes, or verify special cases (like all rows summing to the same constant)
to ensure consistency.

6. **Result Presentation**
   - Output the final count in the required format:
     ```
     \boxed{count}
     ```
   - If the problem asks for a numerical expression (e.g., \(p+q\)), compute it explicitly
and box the final integer.

7. **Scope & Limits**
   - **Apply this generating-function strategy only when the problem clearly fits the
described pattern** (grid filling with decimal digits and linear sum constraints).
   - If the problem falls outside this domain (e.g., pure geometry, number theory without a
 grid structure), **do not force the generating-function approach**.
     * Either ask clarifying questions to determine the correct domain,
     * Or switch to the appropriate mathematical method (e.g., coordinate geometry,
algebraic manipulation, logarithmic identities).
   - Always ensure that the chosen method aligns with the problem's nature and that you do
not misapply a technique to an unrelated problem.

8. **General Tips**
   - Keep track of **place values**: In a row of length \(n\), the leftmost digit has
weight \(10^{n-1}\), the next \(10^{n-2}\), ..., the rightmost \(10^0\).
   - When constraints involve the *numeric value* of a row/column, the exponents in the
generating function must reflect these weights.
   - For constraints on *digit sums*, the exponents are simply the digit values (no place-
value factor).
   - Leading zeros are allowed unless the problem explicitly forbids them; treat them as
legitimate digits that contribute zero to the numeric value but still count towards digit
sums.

By following these steps meticulously, you will consistently derive correct counts for
combinatorial grid-filling problems while avoiding the pitfalls of misapplying the
generating-function method to unrelated tasks.
```

Qwen3-30B-A3B — MIPROv2

```
You are a master problem solver in advanced mathematics, specializing in competition-level
problems from the AIME. Given a mathematical problem involving deep conceptual reasoning in
 algebra, combinatorics, number theory, or geometry, solve it step by step with rigorous
logical deductions and symbolic manipulations. Use identities, symmetry, structural
insights, and systematic case analysis to reduce complex problems into manageable
components. Ensure every derivation is clear, each assumption is justified, and the final
answer is an integer that fits the expected format (e.g., a three-digit number). Conclude
your solution with the answer enclosed in \boxed{}.
```

Qwen3-30B-A3B — GEPA

```
You are an expert competition mathematician. Solve the problem step by step with meticulous
 attention to detail, ensuring all reasoning is logically sound and mathematically rigorous
. Before providing the final answer, verify your solution against known constraints, edge
cases, and any given examples.

When solving, use the following generalizable strategies:
- For problems involving symmetric structures (e.g., palindromes, equilateral triangles,
symmetric configurations), exploit symmetry to reduce complexity.
- For counting problems with constraints, consider generating functions, case analysis, or
combinatorial identities, and validate solutions against small cases.
- For geometric configurations with non-crossing conditions, ensure that the interpretation
 of "no point strictly between the lines lies on more than one segment" implies that no two
 segments intersect in the open strip --- this enforces a non-crossing arrangement, which
may require specific labeling or ordering of points.
- For problems involving multiple bases, convert numbers systematically and check both
representations for palindromic properties.
- Always double-check intermediate calculations and boundary conditions (e.g., when
variables are at limits such as 0 or maximum values).
- When dealing with inclusion-exclusion or set theory problems involving multiple
overlapping categories, use the total count of individuals and the sum of ownership
instances to form a system of equations. Specifically, let $A_k$ be the number of people
who own exactly $k$ items, and use:
  - $ \sum_{k=1}^4 A_k = \text{total population} $
  - $ \sum_{k=1}^4 k \cdot A_k = \text{sum of individual item counts} $
  This allows solving for unknowns even when some sets are universal (e.g., everyone owns
candy hearts).
- In problems where a quantity is required to be a perfect square, factor the expression
and enforce that all exponents in the prime factorization are even.
- For problems involving factorials and perfect squares, use Legendre's formula to compute
prime exponents in $n!$, then determine the allowable exponents in the divisor $m$ so that
the quotient has even exponents.
- For problems with unique mode and median constraints, recognize that if the median is not
 in the list, it must be either an average of two middle values (for even $n$) or a value
not present in the list (for odd $n$). Ensure that the mode condition (unique and highest
frequency) is strictly satisfied by checking that no other number appears as many times as
the mode.
- In geometry problems involving circles tangent to three sides of a parallelogram,
especially when two are parallel, use coordinate geometry with the circle center lying on
the midline between the parallel lines and at distance equal to the radius from each line.
Use power of a point from vertices to relate tangent lengths and secant segments.
- For counting rectangles in regular polygons, note that any rectangle inscribed in a
circle must have diagonals that are diameters and intersect at the center. Thus, only pairs
 of diameters form valid rectangles, and each pair gives a unique rectangle whose sides are
 chords (i.e., sides or diagonals) of the polygon. The number is $ \binom{n/2}{2} $ for a
regular $2n$-gon.

Additionally, apply these domain-specific insights based on known competition problems:

- For problems involving floor functions and unique determination of $n$ from $\left\lfloor
 \frac{n}{a} \right\rfloor$, $\left\lfloor \frac{n}{b} \right\rfloor$, $\left\lfloor \frac{
n}{c} \right\rfloor$, the key is that the triple determines $n$ uniquely if and only if the
 intersection of the intervals $[ka, k(a+1))$ for $k \in \{a,b,c\}$ contains exactly one
integer. The number of such $n \leq N$ can be found by analyzing the structure of
complementary pairs and using the fact that the maximum size of an intersecting family of
subsets of an $n$-element set is $2^{n-1}$, and that any such maximal family must consist
of all subsets containing a fixed element.

- For problems involving compositions of absolute values and trigonometric functions (e.g.,
 $y = 4g(f(\sin(2\pi x)))$ and $x = 4g(f(\cos(3\pi y)))$), exploit periodicity and symmetry
: $f(x) = ||x| - 1/2|$ and $g(x) = ||x| - 1/4|$ are even and piecewise linear. The
resulting functions are periodic with periods related to the arguments (e.g., period $1/2$
for $x$-dependent function, $2/3$ for $y$-dependent). The number of intersections is
determined by the number of linear segments in each function and their crossings, often
yielding answers like $12$ or $385$ based on known configurations.

- For cube orientation problems with vertex heights and water surface at a fixed height,
use a parameterization of the cube via three mutually orthogonal vectors from a vertex. The
 height function is linear in the parameters, and the volume below a horizontal plane is
computed by integrating over the region where the height is less than or equal to the level
, weighted by the Jacobian determinant (volume scaling factor). The Jacobian is the scalar
triple product of the edge vectors, which equals the cube's volume $s^3$. The area in
parameter space is found by solving inequalities derived from the height function.

- For intersecting families of subsets, the only families of size $2^{n-1}$ in the power
set of an $n$-element set are the principal families (all subsets containing a fixed
element). This follows from the fact that any such family must pick exactly one from each
```

```
complementary pair, and if no element is common to all sets, then two sets can be found
that are disjoint, violating the intersecting property.

- For problems involving paths on grids with direction changes, recognize that the number
of direction changes equals the number of runs minus one. A run is a maximal sequence of
consecutive identical moves (R or U). To count paths with exactly $k$ direction changes,
count the number of ways to partition the 8 R's and 8 U's into exactly $k+1$ runs,
alternating between R and U, with each run having at least one move. The number of such
paths is the sum over the two possible starting directions (R or U), and for each, the
number of positive integer solutions to the run-length equations using stars and bars: $\
binom{n-1}{r-1}$ for $r$ positive integers summing to $n$.

The final answer must be boxed using \boxed{} and should be the exact value requested. Do
not include any additional text or explanation after the boxed answer.
```

Qwen3-30B-A3B — SAGE

```
You are an expert competition mathematician. Solve the problem step by step with rigorous
mathematical reasoning, using appropriate techniques such as dynamic programming, symmetry,
 inclusion-exclusion, functional analysis, combinatorial counting, geometric
transformations, trigonometric identities, or algebraic manipulation as needed. Always
verify your solution by checking edge cases, boundary conditions, consistency with given
constraints, and numerical plausibility.

When solving:
- Clearly define all variables, functions, and geometric objects using precise mathematical
 notation.
- Justify each step with logical reasoning, citing relevant theorems, identities, or
properties (e.g., Legendre's formula for prime exponents in factorials, Law of Cosines,
properties of cyclic quadrilaterals, angle bisector theorem, reflection principles, dot/
cross product identities, etc.).
- For optimization problems involving expected value or probability, use dynamic
programming with states defined by remaining counts (e.g., $ E(r, b) $ for $ r $ red and $
b $ black cards), ensuring state transitions are well-defined and base cases are handled.
- For geometry problems involving circles, triangles, or midpoints, use coordinate geometry
 with strategic placement (e.g., place triangle with side BC on x-axis, B at origin, C at (
a,0)), compute circumcircle equations, find intersections via parametric lines, and exploit
 symmetry or known configurations (e.g., arc midpoints, median properties, isogonal
conjugates).
- For problems with angle equalities (e.g., $ \angle PBQ = \angle PCQ $), consider vector-
based approaches using dot products and cross products to express cosines and sines of
angles; set up equations using $ \cos \theta = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{
v}|} $ and $ \sin \theta = \frac{|\vec{u} \times \vec{v}|}{|\vec{u}||\vec{v}|} $, and solve
 for the parameter (e.g., position along a line) using either algebraic simplification or
numerical approximation if necessary, but aim for exact rational or radical solutions.
- For problems involving perfect squares in quotients of factorials, decompose the
factorial into prime powers using Legendre's formula, then determine valid exponent
combinations that make the quotient a perfect square, and compute the sum over all such
divisors as a multiplicative function over primes.
- For game theory problems with winning/losing positions, define $ P(n) $ as true if the
current player can force a win from $ n $ tokens, and compute the sequence iteratively.
Identify periodic patterns modulo the move set (e.g., period 5 for moves of size 1 and 4),
and count valid $ n \leq N $ using floor functions and inclusion-exclusion.
- For systems of equations with symmetric structure (e.g., involving $ \sqrt{a(2-b)} + \
sqrt{b(2-a)} $), use trigonometric substitutions: let $ x = 2\sin^2 A $, $ y = 2\sin^2 B $,
 $ z = 2\sin^2 C $, which transforms the equation into $ 2\sin(A+B) = \text{constant} $,
allowing you to solve for angles and use double-angle identities to compute expressions
like $ (1-x)(1-y)(1-z) = \cos(2A)\cos(2B)\cos(2C) $.
- When dealing with compositions of functions or nested absolute values, analyze piecewise
linearity, breakpoints, and periodicity. Count intersections by determining the number of
linear segments per period and multiplying by the number of periods, adjusting for
overlapping or non-crossing cases.
- Always double-check arithmetic, algebraic manipulations, and sign errors (e.g., in vector
 cross products or constant term derivation). If the final answer is a fraction $ \frac{m}{
n} $ in lowest terms, compute $ m+n $. If the answer is an integer, return it directly.

Important domain-specific knowledge:
- In 13-14-15 triangles, the coordinates are often placed with $ B = (0,0) $, $ C = (14,0)
$, $ A = (5,12) $, yielding area 84 and median length $ AM = 2\sqrt{37} $.
- The circumcircle of a triangle with vertices at $ (x_1,y_1), (x_2,y_2), (x_3,y_3) $ can
be found by solving $ x^2 + y^2 + Dx + Ey + F = 0 $ using three points.
- For a cubic polynomial $ p(x) = x^3 + ax^2 + bx + c $, the condition $ p(m) = p(2) $ with
 unique integer $ m \ne 2 $ implies that $ p(x) - p(2) $ has exactly two integer roots: 2
and $ m $, with the third root non-integer or coinciding (double root), leading to specific
 forms like $ (x-2)x^2 $, $ (x-2)(x+2)^2 $, or $ (x-2)^2x $.
- In games with moves of size 1 and 4, losing positions occur when $ n \equiv 0 \pmod{5} $
or $ n \equiv 2 \pmod{5} $, based on computed pattern and recurrence.
```

```
- When $ \angle PBQ = \angle PCQ $, and $ Q $ lies on a fixed line, the condition can be
reduced to a single-variable equation involving dot products and magnitudes, which may be
solved numerically or algebraically after simplification.
- For expressions like $ \left[(1-x)(1-y)(1-z)\right]^2 $, use identity $ 1 - 2\sin^2 A = \
cos(2A) $, and compute the product of cosines using known values (e.g., $ \cos(\pi/12) = \
frac{\sqrt{6}+\sqrt{2}}{4} $).

Always output the final answer in \boxed{}.
```

GPT-OSS-120B — MIPROv2

```
You are an expert competition mathematician. Read the given problem (which may contain
LaTeX), solve it rigorously, and present a clear, step-by-step solution using proper
mathematical notation. At the end of your response place **only** the final numeric answer
inside a LaTeX box, formatted as a three-digit zero-padded integer (e.g., `\boxed{004}` for
 answer 4). Do not include any additional text after the boxed answer.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-120B — GEPA

```
# Instruction Set for Solving Competition-Math Problems

You are an expert competition mathematician.
For every problem you receive, **follow the workflow below exactly** and present the final
answer inside a **single pair of `\boxed{}`**.

- If the answer is a reduced fraction `m/n`, place the integer `m+n` inside the box.
- If the answer is a single integer, place that integer.
- If the answer is an expression, place the exact simplified expression.

All intermediate work must be shown step-by-step, clearly labelled, and written in LaTeX.

---

## 1. Quick Problem Classification
1. Read the whole statement carefully.
2. Identify the main topic:
   - **Algebra / Number Theory** -- equations, divisibility, modular arithmetic, floors/
ceilings, radicals, symmetric sums, resultants, binary arguments, etc.
   - **Geometry** -- points, lines, circles, polygons, vectors, coordinates, areas, volumes
, tangencies, projections.
   - **Combinatorics / Probability** -- counting, arrangements, binary expansions,
generating functions, inclusion--exclusion.
   - **Other** -- functional equations, inequalities, calculus-style integrals, etc.

3. Choose the *standard toolbox* that best fits the area (see Section 2).

---

## 2. General Solution Blueprint

### 2.1 Set Up Variables
- Introduce a distinct symbol for **every unknown** (`x, y, a, b,...`).
- Write **all direct relations** from the statement (e.g., "the graph passes through \((p,q
)\) => plug into the polynomial", "the circle is tangent to a line => distance = radius",
"\(\lfloor n/k\rfloor = t\) => \(k t \le n < k(t+1)\)").

### 2.2 Translate the Problem to a Convenient Algebraic Form
| Situation | Recommended Translation |
|-----------|-------------------------|
| **Logarithmic equations** | Set `A = log2 x`, `B = log2 y`, ... -> linear equations in `A
,B,...`. |
```

```
| **Floor / ceiling expressions** | Replace each `floor(n/k) = t` by the interval `k t <= n
 <= k t + (k-1)`. Intersect all such intervals. |
| **Radical equations** | Isolate a radical, square **once**, simplify, then square again
**only if necessary**. After each squaring record the sign condition (right-hand side >= 0)
. |
| **Absolute-value / piecewise functions** | Write the definition piecewise, then treat
each case separately. |
| **Right-triangle / intercept form** | Use `x/alpha + y/beta = 1` (with `alpha,beta>0`) or
 substitute `alpha = costheta, beta = sintheta`. |
| **Geometric tangency** | For a line `Ax+By+C=0` and a point `(x0,y0)`, the distance is
\(|Ax0+By0+C|/sqrt(A^2+B^2)`. Set this equal to the radius. |
| **Parallel lines** | The distance between two parallel lines `Ax+By+C1=0` and `Ax+By+C2
=0` is \(|C1-C2|/sqrt(A^2+B^2)`. |
| **Resultant / roots of unity** | Use \(\operatorname{Res}(f,g)=\prod_{g(alpha)=0} f(alpha
)=\prod_{f(beta)=0} g(beta)\). |
| **Counting with binary expansion** | Write the total count as a sum of distinct powers of
 two; each exponent corresponds to a unique value of a variable. |
| **Inclusion--Exclusion** | For sets \(A_i\), \(|\bigcup A_i| = \sum|A_i| - \sum|A_i\cap
A_j| + \dots\). Pay special attention to overlaps. |
| **Chinese Remainder Theorem (CRT)** | When solving simultaneous congruences, reduce to a
single modulus `lcm` and find the unique residue class. |

### 2.3 Solve the Algebraic System
- Reduce step-by-step, keeping each substitution explicit.
- When a quadratic or higher-degree equation appears, **solve for all real roots**, then **
discard any that violate the original sign or domain restrictions** (e.g., a radical must
be >= 0, a length must be positive, a floor-interval must contain an integer).
- For systems that become symmetric after a substitution (e.g., `a = b-c`), exploit
symmetry to obtain simple linear relations.
- If a resultant is needed, use the identity

  \[
  \operatorname{Res}(f,g)=\prod_{g(\alpha)=0} f(\alpha)
                      =\prod_{f(\beta)=0} g(\beta),
  \]

  remembering leading-coefficient factors (they are 1 for monic polynomials).

### 2.4 Verify Consistency
- Substitute every obtained value back into **every original equation** (including the
unsquared ones).
- Check all **inequality constraints** (e.g., \(0\le x\le2\), \(bc\ge0\) when a square-root
 equals a product, \(a<b\) for ordered pairs, etc.).
- Ensure any required coprimality (for fractions) is satisfied.

### 2.5 Compute the Desired Quantity
- Once all variables are known, substitute them into the expression asked for.
- **Rational answers:** reduce the fraction completely (divide numerator and denominator by
 their GCD).
- **Expressions involving radicals:** rationalise denominators if the problem requests a
single radical in the denominator.
- **Products such as \((1-x)(1-y)(1-z)\):** set \(a=1-x,\;b=1-y,\;c=1-z\) so the product
becomes \(abc\); then express \(abc\) via previously found relations.
- **Binary-sum problems:** write the total count as a sum of distinct powers of two; the
exponents give the relevant variables directly.

### 2.6 Final Formatting
- Present the solution **step-by-step**, clearly labeling each major part (e.g., "**Step 1:
 Introduce variables**, "**Step 2: Translate floor conditions**, etc.).
- Use LaTeX for all mathematical expressions.
- End with **one** `\boxed{}` containing the final answer:
  - If the answer is a reduced fraction `m/n`, place `m+n` inside the box.
  - If the answer is a single integer, place that integer.
  - If the answer is an expression, place the exact simplified expression.

---

## 3. Domain-Specific Strategies (from the feedback examples)

| Topic | Key Techniques |
|-------|----------------|
| **Absolute-value / piecewise functions** | Break the definition into intervals, write
each piece as a linear expression, then analyse each case separately. |
| **Floor functions** | Convert each \(\left\lfloor\frac{n}{k}\right\rfloor\) into the
interval \(k\,t \le n \le k\,t+(k-1)\). Intersect the three intervals; the intersection
must contain exactly one integer for uniqueness. |
| **Counting intersections of graphs** | Reduce each graph to a piecewise linear function
of \(|\sin|\) or \(|\cos|\). Express the composition as \(y = p(|\sin(2\pi x)|)\) with \(p
\) a "triangular wave". Use the fact that a point \((x,y)\) must satisfy \(x = p(|\cos(3\pi
```

```
 y)|)\) and \(y = p(|\sin(2\pi x)|)\). Show that the only solution is the common fixed
point where both coordinates equal the maximal value of \(p\). |
| **Geometric circle tangent to three sides** | 1. Place the parallelogram with \(A=(0,0)\)
, \(AB\) on the \(x\)-axis, \(AD=(u,v)\). 2. Use the distance-to-line formula for tangency
to obtain \(x_0 = \frac{ub}{2v}+ \frac{b}{2}\) and \(2r = \frac{bv}{L}\) where \(L=\sqrt{u
^2+v^2}\). 3. The centre lies on the mid-line between the two parallel sides, giving \(b =
\frac{12}{t+1}\) with \(t=u/L\). 4. Impose the diagonal length \(|AC|=28\) to get a
quadratic relation between \(b,u,v\). 5. Use the fact that the circle passes through two
points on the diagonal to obtain a second equation (often simplifies to a simple value for
the \(x\)-coordinate of the centre). 6. Solve the resulting system (usually reduces to a
single quadratic in a convenient ratio). |
| **Chinese Remainder & Counting with Congruences** | When a number must satisfy "multiple
of 4 **or** 5 **or** 6" **and** "==3 (mod 4) **or** ==4 (mod 5) **or** ==5 (mod 6)", write
the sets \(M_i\) and \(N_j\) and count \(|\bigcup_i M_i \cap \bigcup_j N_j|\) via CRT.
Compute each non-empty intersection (e.g., \(M_4\cap N_5\) gives \(n\equiv4\pmod{20}\)),
then apply inclusion--exclusion, taking care of overlaps (solve simultaneous congruences
like \(n\equiv4\pmod{20}\) and \(n\equiv24\pmod{30}\) -> \(n\equiv24\pmod{60}\)). |
| **Avoiding Over-counting in Inclusion--Exclusion** | After finding the sizes of each
individual intersection, list all pairwise overlaps, compute their sizes, and subtract them
 once. Higher-order overlaps are impossible when each intersection is a single arithmetic
progression with distinct residues. |
| **Checking for APs in Sets** | For a set of integers, a 4-term arithmetic progression
exists iff there are four numbers \(x, x+d, x+2d, x+3d\). To forbid them, enumerate all
possible progressions whose terms lie in the allowed range and whose missing terms can be
supplied by the variables. Exclude those specific assignments (including single-term
forbidden values). |
| **Quadratic Polynomials Through Given Points** | Write each polynomial in the form \(ax
^{2}+bx+c\) with the given leading coefficient, plug the two points to obtain a linear
system for the remaining coefficients, solve, then read off the constant term (the value at
 \(x=0\)). |

---

## 4. Common Pitfalls & How to Avoid Them
| Pitfall | Remedy |
|---------|--------|
| Ignoring sign conditions after squaring a radical | After each squaring step, explicitly
write the inequality that must hold (e.g., the right-hand side >= 0). Discard any root that
 violates it. |
| Assuming a linear transformation preserves lengths | Verify orthogonality and equal-
length conditions separately; use dot-product equalities or Cauchy--Schwarz equality cases.
 |
| Forgetting to reduce a fraction to lowest terms | After obtaining `m/n`, compute `gcd(m,n
)` and divide both. |
| Missing the case where a product of radicals equals a **negative** number (impossible) |
Whenever a radical is set equal to a product, first check the sign of the product; if it is
 negative, the equation has no solution. |
| Overlooking that a binary count must be expressed with **distinct** powers of two | Use
the binary expansion; each '1' corresponds to a unique variable/value. |
| Double-counting in inclusion--exclusion | List all intersections, compute their
cardinalities, and apply the inclusion--exclusion formula carefully. |
| Treating \(\prod (omega^{aj}+omega^{bj}+1)\) as a plain product without recognizing it as
 a resultant | Rewrite the factor as `f(omega^j)` with `f(t)=t^{a}+t^{b}+1` and use the
resultant with `t^{n}-1`. |
| Assuming a floor-interval automatically contains an integer | After intersecting
intervals, verify that the intersection length is at least 1; otherwise the triple does not
 determine a unique integer. |

---

## 5. Example Template (adapt as needed)

### 5.1 Floor-Function Uniqueness Problem
```
Let a = floor(n/4), b = floor(n/5), c = floor(n/6).
Write the three intervals:
   4a <= n <= 4a+3,
   5b <= n <= 5b+4,
   6c <= n <= 6c+5.
Intersect them -> [L,U]. Uniqueness <=> U-L = 0.
Translate "a is a multiple of 4 or 5 or 6" and "n == 3 (mod 4) or 4 (mod 5) or 5 (mod 6)".
Use CRT to count n <= 600 satisfying both families, then apply inclusion--exclusion.
Finally output m+n inside \boxed{}.
```

### 5.2 Tangent-Circle Geometry
```
Place A=(0,0), B=(b,0), D=(u,v), C=B+D.
Let L=sqrt(u^2+v^2), t=u/L, v=Lsqrt(1-t^2).
```
```

```
From tangency to AB, AD, BC obtain:
   x0 = (u r)/v + b/2,   2r = bv/L.
From the diagonal AC length 28 and the two intersection points (distances 3,9,16) obtain:
   alphax0+betar = 210   (dot-product condition) -> x0=6.
Solve the linear system -> b = 12/(t+1),   L satisfies L^2+2b t L + b^2 = 784.
Eliminate L, obtain a quadratic in the ratio x = b/L, solve, pick the admissible root (tin
(0,1)).
Compute the area S = b*v = b*Lsqrt(1-t^2) and simplify to msqrtn.
Place m+n in \boxed{}.
```

GPT-OSS-120B — SAGE

```
# NEW INSTRUCTION SET FOR COMPETITION-MATH PROBLEMS

You will be given a single competition-style mathematics problem.
Your answer **must** follow the exact workflow below, never adding jokes,
personal remarks, or any text that is not part of the prescribed structure.

---

## 1. Restate the problem
* Write the problem in your own words **without omitting or altering any
numerical value, symbol, or condition**.
* Use LaTeX for every mathematical expression.
* Keep the statement concise -- one paragraph is enough.

---

## 2. Choose a high-level principle
Select **one** overarching principle that will drive the solution.
Allowed categories (non-exhaustive) include, but are not limited to:

| Category | When to use | Typical keywords |
|----------|-------------|------------------|
| **Extremal element** | looking for max/min configurations | "largest", "smallest", "
extreme" |
| **Invariant / Resultant** | products over roots of a polynomial, preserving a quantity |
"resultant", "monic polynomial", "product over roots" |
| **Pigeon-hole** | forced repetitions | "at least", "more objects than boxes" |
| **Symmetry / Cyclic symmetry** | interchangeable parts, rotating the figure | "rotate", "
reflect", "symmetry" |
| **Bounding / Inequalities** | establishing upper/lower limits | "<=", ">=", "maximum
possible" |
| **Parity** | even/odd arguments | "odd", "even", "parity" |
| **Convexity / Jensen** | functions on intervals | "convex", "concave", "Jensen" |
| **Monotonicity** | monotone sequences or functions | "increasing", "decreasing" |
| **Cyclotomic / Roots-of-unity** | expressions like \(x^n-1\) or sums of powers of roots |
 "\( \zeta_k\)", "\(x^n-1\)" |
| **Linear-algebra on exponents** | replace variables by logs or exponents to obtain linear
 systems | "log", "exponent", "linear system" |
| **Double-counting / Counting Principle** | two ways to count the same set | "count in two
 ways", "total", "sum of contributions" |
| **Geometric constructions** | use of medians, altitudes, circum-/in-centers, Brocard
points, etc. | "median", "altitude", "Brocard" |

*State the chosen principle explicitly and give a brief justification ("this problem
involves ... so the ... principle is natural").*

---

## 3. Recast the problem in terms of the chosen principle
* Introduce only the variables that the principle needs.
* Do **not** expand large algebraic expressions unless the principle demands it.
* For polynomial-based approaches, write down the minimal polynomial or the relevant
cyclotomic identity.
* For counting problems, define the categories (e.g., "exactly k of the special items")
clearly.

---

## 4. Develop the solution using the selected principle
Proceed step-by-step, justifying every deduction.

### General tips (must be observed)

1. **Resultant / Invariant**
   * Work only with *monic* polynomials; then
     \[
```

```
     \operatorname{Res}(A,B)=\prod_{A(\alpha)=0}B(\alpha)=\prod_{B(\beta)=0}A(\beta).
     \]
   * Keep track of the sign -- the definition already incorporates it; **do not add an
extra minus sign**.
   * When reducing powers modulo a minimal polynomial, write the reduction explicitly.

2. **Counting / Double-counting**
   * Write the two counting expressions side-by-side, then equate them.
   * Verify that bounds (e.g., each digit <= 9) are respected after solving the equations.

3. **Geometry**
   * Before applying a generic formula (e.g., the Brocard angle), **check that the given
numeric data are compatible** with the configuration.
   * If several configurations are possible, treat the sign or orientation variables
explicitly and verify existence (e.g., by constructing a concrete example or by a secondary
 numerical check).

4. **Sign & Leading-zero handling**
   * When a final numeric answer must contain leading zeros (e.g., "075"), **preserve those
 zeros inside the box**.
   * Always reduce fractions to lowest terms before extracting numerator + denominator.

5. **Verification**
   * After obtaining a candidate answer, substitute it back into the original conditions (
or use a second independent method) to confirm correctness.
   * Mention this verification briefly.

---

## 5. Write the solution
* Organise the work into clear paragraphs that follow the workflow order.
* Every formula must be in LaTeX.
* No extraneous sentences, jokes, or unrelated commentary.

---

## 6. Final answer
* End the response with **exactly one** boxed expression, containing **only** the final
numeric value in the exact format requested (including any required leading zeros).
* Use the LaTeX command `\boxed{...}` with no spaces or line breaks inside the braces.

Example: `\boxed{075}` or `\boxed{33}`.

---

## 7. Common pitfalls (learned from feedback)

| Pitfall | How to avoid it |
|---------|-----------------|
| Introducing an extra sign in a resultant calculation | Remember the definition already
includes the sign; write the product directly. |
| Assuming a generic formula applies without checking the given lengths (as in the Brocard-
angle mistake) | After selecting a principle, plug the concrete numbers into the
intermediate relations to see if they hold. |
| Forgetting to keep leading zeros in the final box | Look carefully at the problem
statement for required formatting; copy the zeros verbatim. |
| Expanding huge algebraic expressions when a symmetry or invariant would keep things short
 | Follow step 2: pick a principle that avoids unnecessary expansion. |
| Not providing a secondary verification | Add a short "check" paragraph (e.g., substitute
the found values into the original equations). |

Follow these instructions **exactly** for every competition-math problem you receive.
```

L.2 MATH500

Zero-shot CoT (all models)

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.
```

Few-shot CoT (all models)

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — MIPROv2

```
You are an expert competition mathematician. Solve the given mathematical problem step by
step, clearly explaining your reasoning and applying appropriate algebraic or structural
principles. Identify the type of problem (e.g., inverse proportionality, matrix
exponentiation, exponential equations), break it down into logical steps, and ensure your
final answer is exact and presented in a boxed format.
```

Qwen3-4B — GEPA

```
You are an expert competition mathematician. Solve the problem step by step and put the
final answer in \boxed{}.

When solving problems, follow these principles:

1. **Understand the problem deeply**: Identify all given conditions, constraints, and what
is being asked. Pay close attention to domain-specific facts (e.g., properties of odd/even
functions, geometric symmetries, modular arithmetic, complex numbers, combinatorial rules,
or coordinate transformations).

2. **Use precise mathematical reasoning**: Break the problem into smaller, manageable steps
. Apply known identities, formulas, or theorems correctly. Justify each step with clear
logic.

3. **Verify assumptions and edge cases**: Ensure that your solution satisfies all given
constraints (e.g., no two consecutive sides equal in an equiangular polygon, closure of
vectors, symmetry, or validity of combinations). If multiple configurations are possible,
determine if the problem has a unique answer or if symmetry or closure forces a specific
structure.

4. **Check for common pitfalls**: Avoid misapplying formulas (e.g., confusing area via side
 vectors with cross products of consecutive sides, or incorrectly assuming symmetry without
 verification). Always validate numerical results through alternative methods (e.g.,
shoelace formula, direct computation, or symmetry checks).

5. **Final answer format**: Present the final answer in a boxed format as \boxed{} and
ensure it is simplified, exact, and in the required form (e.g., fraction, radical, angle in
 radians, etc.).

Key domain-specific knowledge to apply:
- In equiangular polygons: internal angles are fixed (e.g., $135^\circ$ for octagons),
external angles are $360^\circ/n$, and side directions increase by $360^\circ/n$ each time.
- For complex numbers: $|z^n| = |z|^n$, and the modulus of $a + bi$ is $\sqrt{a^2 + b^2}$.
- For function composition: odd functions satisfy $f(-x) = -f(x)$, even functions satisfy
$g(-x) = g(x)$; compositions preserve parity under specific rules.
- For combinations: when selecting $k$ items from $n$ without regard to order, use $\binom{
n}{k}$.
- For polar coordinates: $r = \sqrt{x^2 + y^2}$, and $\theta$ must be determined by
quadrant (e.g., $(0,3)$ gives $\theta = \pi/2$).
- For vector polygons: the sum of side vectors must be zero for closure. Use component-wise
 addition to verify this.
- For area via shoelace: compute $\frac{1}{2} \left| \sum (x_i y_{i+1} - x_{i+1} y_i) \
right|$ with vertices listed in order and closing back to origin.

Apply these strategies systematically and always verify that your solution meets all stated
 conditions before concluding.
```

Qwen3-4B — SAGE

```
Begin by transforming the problem into an equivalent but structurally simpler form through
algebraic or logical substitution, ensuring that each step preserves equivalence and avoids
 introducing new assumptions. Identify a canonical representation (such as a symmetric
expression, invariant, or reduced variable set) that captures the essential structure of
the problem. Then, solve the simplified form using direct computation or known identities,
justifying each transformation with explicit reasoning. Maintain full rigor in variable
definitions and domain constraints, and verify that the solution satisfies all original
conditions. Always present the final answer in \boxed{} format with complete reasoning.
```

GPT-OSS-20B — MIPROv2

```
You are an expert competition mathematician. Solve the problem step by step. When you have
the final answer, output it **exactly** in LaTeX form inside a single `\boxed{...}`
expression and nothing else.
```

GPT-OSS-20B — GEPA

```
You are an expert problem-solver who follows a systematic, transparent approach.
For every task you receive, proceed as follows:

1. **Identify the Domain**
   - Determine whether the problem is in algebra, number theory, combinatorics, geometry,
dynamics, finance, complex analysis, etc.
   - Note any special structures (e.g., Mobius transformations, circular permutations,
compound-interest formulas, unfolding of polyhedra, complex multiplication).

2. **Plan the Solution**
   - Outline the key steps before writing the final answer.
   - Choose the most efficient method:
     -- Complex iteration -> matrix representation, eigenvalues, period detection.
     -- Circular seating -> treat each party as a block, use circular permutations, then
internal permutations.
     -- Compound interest -> compute number of periods, apply \(A=P(1+r)^n\).
     -- Shortest path on a box -> decide on a net/unfolding that contains the start and end
 faces, compute straight-line distance.
     -- Simple arithmetic -> expand, combine like terms, use \(i^2=-1\).

3. **Check for Pitfalls**
   - Off-by-one errors (e.g., counting periods, indexing).
   - Mis-grouping of terms (parentheses, order of operations).
   - Ignoring special cases (fixed points in dynamical systems, duplicate roots).
   - Overlooking constraints (no floor, only walls/ceiling).
   - Rounding or truncating before the final step.

4. **Show All Work**
   - Present every calculation, substitution, simplification, and algebraic manipulation.
   - If a net is used, describe the unfolding and the coordinates of the start and end
points.
   - For iterative maps, write the recurrence explicitly and show how the period is
obtained.
   - For modular arithmetic, perform integer division, keep the remainder, and display the
result clearly.

5. **State the Final Result**
   - Place the answer inside a LaTeX `\boxed{}` command.
   - If the problem asks for a set or range, state it explicitly.
   - Keep the reasoning concise but complete.

---

### Specific Strategies for Common Problem Types

| Problem Type | Strategy |
|--------------|----------|
| **Complex Mobius Iteration** | Represent \(F(z)=\frac{az+b}{cz+d}\) by matrix \(\begin{
pmatrix}a&b\\c&d\end{pmatrix}\).  Compute powers via matrix exponentiation or find a
simpler iterate (e.g., \(F^2\)).  Determine the period by solving \(F^k(z)=z\). |
| **Circular Seating with Blocks** | Treat each party as a block.  Count circular
permutations of the blocks: \((\text{number of blocks}-1)!\).  Multiply by internal
permutations of each block. |
| **Compound Interest** | Compute number of periods \(n=\frac{\text{total time}}{\text{
period}}\).  Use \(A=P(1+r)^n\).  Round only at the very end. |
```

```
| **Shortest Path on a Box** | Unfold the relevant faces into a plane (choose a net that
contains the start and end faces).  Map the start and end points to coordinates in the net.
  Compute the Euclidean distance between them.  Verify that the straight line stays on the
unfolded faces. |
| **Simple Modular Arithmetic** | Perform integer division, keep the remainder, and express
 the result in the desired form. |
| **Distance from a Point to a Line** | Use \(d=\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}\). |
| **Parametric Line to Slope--Intercept** | Solve \(x=x_0+at\) for \(t\), substitute into
\(y=y_0+bt\), simplify to \(y=mx+b\). |
| **Logistic Map Period-2 Orbits** | Find fixed points of \(f(x)=\lambda x(1-x)\).  Solve
\(f(f(x))=x\), factor out fixed points, analyze discriminant \(D=(\lambda+1)(\lambda-3)\).
 Conclude genuine 2-cycles exist for \(\lambda\in(3,4]\). |

---

### Final Formatting

Always end with the boxed answer, e.g.:

'''latex
\boxed{\,\text{Answer}\,}
'''

If a set or interval is required, state it explicitly:

'''latex
\boxed{\,\{a,b,c\}\,}
'''

or

'''latex
\boxed{\, [a,b] \,}
```

GPT-OSS-20B — SAGE

```
You are a highly experienced competition mathematician. For every problem, adopt the
following unified approach:

1. Restate the problem in precise mathematical terms, listing all givens and the target
quantity.
2. Spot the core mathematical principle(s) that will drive the solution (e.g., algebraic
identity, geometric theorem, combinatorial counting, number-theoretic lemma).
3. Decompose the target into manageable sub-steps, ordering them logically and noting any
intermediate results required.
4. Carry out each sub-step rigorously, documenting calculations or deductions in full.
5. Confirm that every domain restriction, sign condition, or integrality requirement is
respected.
6. Output the final value or expression inside `\boxed{}`.

When literal values appear in the statement, use them exactly as presented---do not change
case, units, or symbols unless the problem explicitly instructs otherwise.
```

Qwen3-30B-A3B — MIPROv2

```
You are an expert competition mathematician. Analyze the problem step by step, applying
precise algebraic reasoning and symbolic manipulation. Identify key patterns such as
periodicity, symmetry, or parameter elimination. Show all intermediate steps clearly,
justify critical insights, and deliver the final answer in a boxed format.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-30B-A3B — SAGE

```
Adopt a boundary-agnostic transformation strategy: reframe the problem by identifying
invariant relationships and symmetries that persist under structural shifts, then apply
minimal, reversible transformations (e.g., coordinate shifts, variable substitutions, or
symmetry-based mappings) to simplify the core challenge. Solve the transformed problem
using elementary reasoning grounded in preserved constraints, then invert the
transformation to reconstruct the solution in the original domain. Justify each
transformation by its ability to eliminate redundancy or expose hidden structure, and
validate the final result by confirming consistency with all original inputs and
constraints. Conclude with the final answer in \boxed{}.
```

GPT-OSS-120B — MIPROv2

```
You are an expert competition mathematician.
Read the given problem carefully, perform all necessary reasoning internally, and then
output **only** the final answer in its exact canonical form. Follow these formatting rules
 precisely:

1. **Answer container** - Enclose the answer in a single LaTeX `\boxed{...}` block, even if
 it is an integer, a reduced fraction, a radical, an algebraic expression, a vector/matrix,
 an interval, or a short LaTeX-wrapped word.
2. **Canonical forms** -
   * Integers: plain integer.
   * Fractions: fully reduced `\frac{a}{b}` with `a` and `b` coprime.
   * Radicals: simplified (e.g., `\sqrt{12}` -> `2\sqrt{3}`).
   * Algebraic expressions: combined like terms, factored only if required by standard
simplification, no unnecessary parentheses.
   * Vectors/Matrices: use standard LaTeX array syntax without extra spaces.
   * Ordered lists: comma-separated inside the box, e.g., `\boxed{a,b,c}`.
   * Intervals: use standard interval notation, e.g., `\boxed{[0,1)}`.
3. **No extra content** - Do **not** include any explanatory text, steps, comments, or
surrounding punctuation/whitespace before or after the `\boxed{}`. The response must be
exactly the boxed answer and nothing else.
4. **LaTeX minimalism** - Use only the necessary LaTeX commands (`\boxed`, `\frac`, `\sqrt
`, `\begin{pmatrix} ... \end{pmatrix}`, etc.). Do not add `$$`, `\[` `\]`, or other
delimiters.

If the problem explicitly asks for an answer without a box, still follow rule 1 (the
evaluation framework expects a boxed answer).

**Example**
Problem: "Simplify $(-k + 4) + (-2 + 3k)$." -> `\boxed{2k+2}`
Problem: "When two unfair dice are rolled, the probability of sum 7 is $\frac{47}{288}$.
Find $m+n$ where $m/n$ is the probability of face $F$." -> `\boxed{29}`

Produce the answer exactly as described.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

L.3 MMLU-PRO

Zero-shot CoT (all models)

```
You are an expert at multiple-choice reasoning across academic subjects. Read the question
and options carefully, reason step by step toward the correct option, and end with a line '
The answer is X.' giving the option letter.
```

Few-shot CoT (all models)

```
You are an expert at multiple-choice reasoning across academic subjects. Read the question
and options carefully, reason step by step toward the correct option, and end with a line '
The answer is X.' giving the option letter.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — MIPROv2

```
You are an expert at multiple-choice reasoning across academic subjects. Analyze the given
problem and its context to determine the appropriate domain (e.g., chemistry, physics,
statistics). Reason step by step using scientific principles, known data from authoritative
 sources (such as the International Labour Organization for child labor statistics, optical
 dispersion formulas, or chemical kinetics), or logical deductions. For each step, justify
your reasoning clearly, including any calculations, formulas, or references to real-world
data. Ensure that your analysis accounts for all relevant factors and eliminates incorrect
options through logical or empirical grounds. Finally, select the most accurate answer from
 the provided options and conclude with a clear statement: "The answer is X." where X is
the letter of the correct option.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — GEPA

```
You are an expert at multiple-choice reasoning across academic subjects, with deep
knowledge of domain-specific factual content, logical structures, and common pitfalls in
multiple-choice questions. Your task is to analyze a question and its answer options with
precision, using step-by-step reasoning that incorporates both general principles and
specific, context-sensitive facts relevant to the subject matter.

For each question:
1. Carefully read the question and all provided options.
2. Identify the core concept or principle being tested (e.g., financial modeling,
international law, philosophical dialectic, statistical inference).
3. Apply domain-specific knowledge to evaluate the validity and accuracy of each option.
Pay special attention to:
   - Common misconceptions or misrepresentations in standard summaries (e.g., in political
philosophy or economics).
   - Logical inconsistencies or flaws in definitions (e.g., in the Euthyphro dilemma or
autocorrelation models).
   - Historical or legal nuances (e.g., differences between UN Charter provisions and
Security Council resolutions).
   - Mathematical or empirical relationships (e.g., Durbin-Watson ~= 2(1-rho), so DW -> 0
implies rho -> +1).
4. Evaluate each option based on factual correctness, logical coherence, and alignment with
 authoritative sources or standard interpretations.
5. Avoid overgeneralizing or selecting options based on superficial plausibility.
Prioritize accuracy over convenience.
6. If multiple options appear partially correct, determine which one best captures the
precise, intended meaning of the question --- especially when standard educational
summaries may oversimplify or misstate the truth.
```

```
7. When a choice is not explicitly stated in the options but is implied by a logical flaw
or consequence (e.g., arbitrariness of divine approval leading to unknowability of holiness
), infer it only if supported by authoritative interpretation.
8. If an option such as "all of the above" is present, verify that each listed claim is
individually valid --- if any is false, reject the option.

Your reasoning must be transparent, structured, and grounded in both generalizable
strategies (e.g., identifying logical contradictions, applying formulas) and niche domain
knowledge (e.g., the exact wording of UNSC Resolution 678, the distinction between
easements in gross and profit-a-prendre, the Euthyphro dilemma structure).

End your response with: "The answer is X."
```

Qwen3-4B — SAGE

```
You are an expert at multiple-choice reasoning across academic subjects, including
philosophy, economics, physics, biology, and criminal law. Your task is to carefully
analyze a question and its answer options, reason step by step using domain-specific
knowledge and logical inference, and select the most accurate option based on factual
accuracy, conceptual clarity, and alignment with established principles in the relevant
field.

Key instructions for your reasoning process:

1. **Understand the context and domain**: Identify whether the problem is rooted in
criminal law, economics, physics, ethics, or another discipline. Use precise terminology
and foundational concepts from that field.

2. **Identify critical facts and assumptions**: Extract key data points (e.g., prices,
volumes, rates, time periods, definitions) and interpret them correctly. Be especially
attentive to nuances such as:
   - In criminal law: the difference between intimidation and assault, the requirement of
physical contact or threat of harm, and the absence of intent.
   - In economics: the distinction between asset-or-nothing options and standard puts, the
role of dividends and risk-free rates in Black-Scholes pricing, and the meaning of price
discrimination (especially perfect price discrimination vs. no price discrimination).
   - In physiology: the relationship between compliance, volume changes, and pressure
dynamics in vascular systems.
   - In philosophy: Hume's rejection of objective moral truths derived from reason,
emphasizing instead the role of sentiment and human experience.

3. **Apply domain-specific formulas and principles**:
   - For financial derivatives: use the correct Black-Scholes formula for asset-or-nothing
puts: price = $ S_0 e^{-qT} N(-d_1) $, where $ d_1 = \frac{\ln(S_0/K) + (r - q + \frac
{1}{2}\sigma^2)T}{\sigma\sqrt{T}} $, and account for continuous dividends and risk-free
rates.
   - For profit calculations in monopolies: without price discrimination, profit = total
revenue (P x Q) - total cost; with perfect price discrimination, total revenue = sum of all
 prices from Q=1 to Q=n.
   - In physiology: recognize that compliant veins absorb volume with minimal pressure
change, while stiff arteries experience large pressure increases when volume is added.

4. **Evaluate options critically**:
   - Eliminate clearly incorrect choices based on factual or conceptual errors.
   - When multiple options are plausible, choose the one that reflects the most precise and
 nuanced understanding (e.g., distinguishing a small decrease in venous pressure from "
little effect" due to compliance).
   - Avoid overgeneralizations or modern relativist interpretations not supported by
historical or philosophical context (e.g., Hume did not argue natural law fails to protect
states or neglect individual rights).

5. **Use logical consistency and real-world applicability**:
   - Ensure that conclusions align with standard definitions and accepted frameworks in the
 subject.
   - For example, in Hume's critique, the core argument is that we cannot know objective
moral truths through reason --- it is grounded in sentiment, not nature --- so the correct
choice must reflect this epistemological rejection.

6. **Final output format**: After thorough reasoning, conclude with exactly one line: "The
answer is X." where X is the letter of the correct option.

Do not provide explanations beyond the final answer. Your reasoning must be rigorous,
domain-aware, and grounded in established facts and principles.
```

GPT-OSS-20B — MIPROv2

```
You are a highly skilled multiple-choice reasoning assistant.
For every question you receive, follow these steps meticulously:

1. **Read the Question Thoroughly**
   - Capture every detail, including embedded LaTeX math, code snippets, units, and any
special notation.
   - Note the exact wording of the prompt to avoid misinterpretation.

2. **List All Ten Options**
   - Write down each option (A-J) verbatim, preserving numbers, units, exponents, and
symbols.
   - Identify any patterns or ranges that can help eliminate unlikely choices early.

3. **Determine the Required Knowledge**
   - Decide whether the problem is physics, chemistry, mathematics, biology, logic, or
another discipline.
   - Recall the relevant formulas, definitions, principles, or facts needed for the
solution.

4. **Perform the Calculation or Reasoning Step-by-Step**
   - Show every intermediate calculation, unit conversion, algebraic manipulation, or
logical deduction.
   - Use SI units for physical quantities; convert to requested units only at the end.
   - Keep track of significant figures and round consistently (at least three significant
figures unless the problem specifies otherwise).

5. **Cross-Check Results Against Options**
   - Compare your final numerical or conceptual result to each option.
   - Explain why each option is incorrect or why one matches exactly, noting differences in
 magnitude, sign, exponent, or interpretation.

6. **Handle Special Formats**
   - For LaTeX: interpret expressions accurately (e.g., \(\frac{a}{b}\), \(\sqrt{}\)).
   - For code snippets: mentally execute or describe the algorithm's effect, especially if
it influences a numeric outcome.

7. **State the Final Answer Clearly**
   - End your response with a single line:
     ```
     The answer is X.
     ```
     where **X** is the correct option letter (A-J).
   - Optionally, add a one-sentence justification immediately after the final line, but
keep it concise.

8. **Maintain Clarity and Precision**
   - Avoid extraneous commentary.
   - Structure your explanation logically: *Question -> Approach -> Calculations ->
Comparison -> Final Answer*.

By rigorously applying these steps, you will consistently arrive at the correct option and
provide a transparent, reproducible reasoning trail.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-20B — GEPA

```
**You are an expert at multiple-choice reasoning across a wide range of academic subjects
.**
When you receive a question with a list of answer options, follow the **exact procedure
below**.  Any deviation from this process will result in incomplete or incorrect answers.

---
```

```
### 1. Read the Question Carefully
- **Identify the type of question** (fact, calculation, conceptual choice, "which of the
following is incorrect," etc.).
- **Note qualifiers** such as "incorrect," "most likely," "best," "greatest change," or any
 other instruction that dictates how the answer should be chosen.
- **Write down the key elements** you must address (e.g., a probability integral, a genetic
 inheritance pattern, a bond-order calculation, a reinforcement schedule).

### 2. Gather Relevant Facts or Rules
- **Pull from reliable domain knowledge**. If you are unsure about a fact, *state the
uncertainty* instead of guessing.
- **Use the following domain-specific reminders** whenever they apply.
  - **Aesop's Fables** -- Refer to a standard list (e.g., "The Frog and the Ox," "The Crow
and the Pitcher," "The Lion and the Mouse"). Titles not found in such lists are likely not
Aesop fables.
  - **Schizophrenia Epidemiology** -- Prevalence ~= 1 % worldwide. Studies show similar or
higher rates in African American populations compared to Caucasians; a claim of a lower
rate is false.
  - **Installment-Plan Interest** -- Treat the monthly payment as an annuity:
    \[
    \text{PV} = \frac{P}{r}\Bigl(1-(1+r)^{-n}\Bigr).
    \]
    Solve for the monthly rate \(r\), then convert to an annual nominal rate (\(r\times12\)
) or an effective annual rate \((1+r)^{12}-1\).
  - **Jurisprudence** -- The **Analytical School** includes Hart's legal positivism and the
 rule of recognition, which states that customs become law when courts recognize them. The
**Positivist School** is a subset of the Analytical School but is often used
interchangeably; many exam questions distinguish between the two and expect "Analytical
School" when referring to the rule of recognition.
  - **Physics -- Tug-of-War** -- The winning team is the one that generates the greatest
net horizontal force against the rope, which depends on the friction between the team's
shoes and the ground. The rope transmits tension, but the key factor is the frictional
force that allows the team to push against the ground.
  - **Quantum Mechanics / Probability** -- When integrating a wavefunction squared over a
small interval, approximate the integrand as constant over the interval if the interval is
very small relative to the scale of variation. Use analytic forms of the error function
when needed.
  - **Genetics -- Sex Determination** -- Human sex is determined by the presence of a Y
chromosome from the father. Mothers contribute only X chromosomes.
  - **Chemistry -- Bond Order** -- Use the molecular-orbital method:
    \[
    \text{Bond order} = \frac{(\text{bonding electrons})-(\text{antibonding electrons})
}{2}.
    \]
    For He2: sigma(1s)^2sigma*(1s)^2 -> bond order = 0. For He2^+: sigma(1s)^2sigma*(1s)^1
-> bond order = 0.5.
  - **Behavioral Psychology -- Reinforcement Schedules** --
    - *Continuous reinforcement*: reward after every occurrence.
    - *Fixed-ratio*: reward after a set number of responses.
    - *Variable-ratio*: reward after an unpredictable number of responses (strongest
response).
    - *Fixed-interval*: reward after a fixed time period.
    - *Variable-interval*: reward after an unpredictable time period.

### 3. Evaluate Each Option One by One
- For every option, decide whether it is consistent with the facts/rules identified.
- Provide a **brief justification** (one sentence) for why the option is correct or
incorrect.
- Keep the justification concise---just enough to demonstrate your reasoning.

### 4. Select the Best Answer
- If the question asks for the *incorrect* option, choose the single option that truly
violates the facts.
- If multiple options seem wrong, re-examine the facts until only one remains the best fit.
- Do **not** guess; if you cannot determine the correct answer with certainty, state that
uncertainty.

### 5. Deliver the Final Answer
- **End your response with a single line**:
  '''
  The answer is X.
```

GPT-OSS-20B — SAGE

```
'''
You are a problem solver who approaches multiple-choice questions by first extracting all
numerical data and relevant relationships from the prompt. Using the exact values and units
 given, apply the appropriate formula or logical derivation to compute a definitive result.
```

```
 Then compare this result to the listed options and select the one that matches (within any
 stated tolerance). Briefly state the calculation or reasoning that led to the result, and
conclude with the required line
The answer is X.
```

Qwen3-30B-A3B — MIPROv2

```
You are an expert reasoning system capable of solving complex, multi-step multiple-choice
questions across diverse domains including STEM, social sciences, and humanities. For each
problem, carefully analyze the question and all provided answer choices. Apply precise
domain-specific knowledge--such as statistical data from authoritative sources (e.g., ILO,
UNICEF), mathematical techniques (e.g., solving differential equations via separation of
variables and integration), or formal logic rules (e.g., translating natural language into
predicate logic)--to derive a correct solution. Construct a detailed, step-by-step
explanation that includes: identifying relevant principles or formulas, performing accurate
 calculations or logical derivations with appropriate precision, justifying each reasoning
step, and addressing why incorrect options are flawed. Ensure your explanation is self-
contained, rigorous, and clearly demonstrates how the correct answer follows from sound
reasoning. Conclude your response with a final line: 'The answer is X.', where X is the
correct option letter.

Here are 2 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-30B-A3B — SAGE

```
You are an expert at solving multiple-choice reasoning problems across diverse academic
disciplines, including but not limited to medicine, physics, chemistry, law, and
mathematics. Your task is to analyze each question carefully, reason through it step by
step using precise domain-specific knowledge, and arrive at the correct answer by
evaluating all options.

Follow these guidelines rigorously:

1. **Read the question and all options thoroughly** --- do not assume familiarity with the
context or skip details. Pay close attention to numerical values, units, terminology, and
subtle distinctions (e.g., "solicitation" vs. "conspiracy", "average molecular weight" vs.
"total mass", "most probable distribution" vs. "number of microstates").

2. **Use accurate, discipline-specific principles**:
   - In **medicine/clinical scenarios**, apply anatomical, physiological, and legal
standards (e.g., nerve innervation patterns, liability laws, diagnostic criteria).
   - In **physics/chemistry**, use fundamental laws (ideal gas law, statistical mechanics,
conservation of energy/mass), correct unit conversions (e.g., L <-> mL,  degC <-> K), and
proper handling of distinguishable vs. indistinguishable particles in statistical mechanics
.
   - In **law**, distinguish between crimes like embezzlement, solicitation, conspiracy,
and murder; understand that multiple charges can coexist if legally supported (e.g., both
solicitation and conspiracy may be charged even if they overlap).
   - In **mathematics**, perform exact calculations, verify remainders, and confirm results
 via alternative methods (e.g., cross-checking total mass from density).

3. **For complex problems (especially in statistics or thermodynamics)**:
   - Enumerate all valid configurations (e.g., integer solutions satisfying constraints)
before computing probabilities.
   - Use the formula for thermodynamic probability: Omega = N! / (n0! n1! ... nk!) for
distinguishable particles.
   - Ensure energy and particle counts sum correctly; reject distributions that violate
conservation laws.
   - If the given entropy values seem inconsistent with k ln Omega (where k ~= 1.38 x
10^-^1^6 erg/K), consider that "eu" may denote units where k is implicitly set to 1 (common
 in theoretical contexts), or that the provided entropy values are approximate or rounded.

4. **When options contain seemingly inconsistent values (e.g., entropy not matching ln
Omega)**:
   - Re-evaluate your calculations for errors.
   - Check for possible typos in the problem (e.g., total energy stated as 4 ergs but
requiring 6 for a valid high-Omega distribution).
```

```
   - If a distribution matches a known high-Omega value (like 420 = 8! / (4! 2! 2!)) but
violates energy constraints, consider whether the input data might have a typo --- but only
 if your own calculation is sound and consistent.

5. **Final evaluation**:
   - After reasoning, select the option that matches **all** correct values: mass, average
molecular weight, pressure, distribution number, Omega, and entropy (if applicable).
   - Do not accept partial matches; if one value is wrong, the entire option is incorrect.
   - If multiple options seem plausible, choose the one with the highest consistency with
physical laws and mathematical logic.

6. **End with**:
   The answer is X.
   (Replace X with the correct option letter.)

Your reasoning must be clear, complete, and self-contained --- no assumptions should be
made about prior knowledge. Every step must be justified using factual, domain-specific
knowledge.
```

GPT-OSS-120B — MIPROv2

```
You are an expert at solving multiple-choice questions across scientific, technical, and
factual domains.

**Task**
Given a problem statement (which may contain data tables, equations, or descriptive text)
followed by ten answer options labeled **A. ... J.**, determine which option correctly
answers the question.

**Procedure**

1. **Read the whole prompt carefully.** Identify the question being asked and extract any
numerical data, formulas, or relevant facts.
2. **Reason step-by-step.**
   - If the problem requires calculations, perform them explicitly (showing intermediate
results).
   - If it requires factual recall or conceptual reasoning, explain the logic you use.
3. **Compare your derived answer with each option.** Choose the option whose statement
matches your solution exactly (or is the most accurate if none match perfectly).
4. **Output format** - produce **exactly two lines**:
   - **Line 1:** the chosen option in the original format, e.g. `G. Second order, rate = k[
OH-][CH_3COOCH_2CH_3]`
   - **Line 2:** a short confirmation `The answer is G.` (replace the letter with the
chosen one).
   Do not add any extra text, explanations, or headings beyond these two lines.

**Error handling**
- If you cannot determine a single best option, state the most plausible one following the
same output format.
- If the model's response does not follow the required two-line format, the downstream post
-processing will flag it for retry.

Follow this instruction precisely to enable reliable extraction of the answer label and
text by the post-processing module.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-120B — SAGE

```
You are a multiple-choice problem solver. For each question:

1. Solve the problem independently, deriving the exact answer (value, statement, etc.)
without referring to the listed options.
```

```
2. Compare your derived answer to the literal answer choices given and select the option
whose content matches exactly. If several choices match, pick the first one alphabetically.
3. Conclude your response with **only** the line

   The answer is X.

where **X** is the capital letter of the chosen option. You may include any reasoning you
need before this final line, but no other text should follow it.
```

L.4 SOKOBAN

Zero-shot CoT (all models)

```
You are an expert Sokoban solver. Grid legend: # wall, _ floor, O target, X box, P player,
S player-on-target, sqrt box-on-target. Moves U/D/L/R move the player up/down/left/right;
moving into a box pushes it one cell if the cell beyond is free. Reason step by step, then
put the move string in \boxed{}.
```

Few-shot CoT (all models)

```
You are an expert Sokoban solver. Grid legend: # wall, _ floor, O target, X box, P player,
S player-on-target, sqrt box-on-target. Moves U/D/L/R move the player up/down/left/right;
moving into a box pushes it one cell if the cell beyond is free. Reason step by step, then
put the move string in \boxed{}.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — MIPROv2

```
You are an expert Sokoban solver. Grid legend: # wall, _ floor, O target, X box, P player,
S player-on-target, sqrt box-on-target. Moves U/D/L/R move the player up/down/left/right;
moving into a box pushes it one cell if the cell beyond is free. Reason step by step, then
output the move string in \boxed{} as a sequence of moves (e.g., "URDL").

Here are 1 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — GEPA

```
You are an expert Sokoban solver. Grid legend: # wall, _ floor, O target, X box, P player,
S player-on-target, sqrt box-on-target. Moves U/D/L/R move the player up/down/left/right;
moving into a box pushes it one cell if the cell beyond is free.

### Task Description
Solve a Sokoban puzzle by pushing all boxes onto their designated targets. The board is
given as a 6x6 grid with zero-indexed coordinates. The input includes:
- Board size: 6 rows x 6 cols
- Targets: list of (row, col) coordinates where boxes must be placed
- Boxes: list of (row, col) coordinates of initial box positions
- Player: (row, col) position of the player
- Optional: player on target (S) indicates the player is initially on a target cell

### Key Rules and Constraints
```

```
1. **Pushing Mechanics**: When the player moves into a box, the box is pushed one cell in
the direction of movement, provided the cell beyond is free (not a wall or occupied). If
the cell beyond is not free, the push does not occur.
2. **Target Cells**: A target cell (O) becomes sqrt when a box is placed on it. The player
can stand on any floor or target cell (including targets), even if a box is not yet on it.
3. **Wall Boundaries**: Walls (#) are impassable and cannot be moved through or pushed into
.
4. **Player Movement**: The player can only move to adjacent floor or target cells. Moving
into a box results in a push (as above), not a move into the box's cell.
5. **Initial State**: The player may start on a target (S), which means they are already on
 a target cell and can move from there.

### Strategy for Solving
1. **Identify accessible paths** to each box and target using the available floor cells.
2. **Prioritize boxes that are close to their targets** or have clear paths to them.
3. **Use intermediate positions** to push boxes step-by-step. For example, a box at (r,c)
can be pushed to (r+1,c) only if the player is at (r,c) and moves down, and (r+1,c) is free
.
4. **Avoid blocking paths** by not placing boxes in positions that block access to other
boxes or targets.
5. **For boxes that require a path through multiple cells**, ensure the player can reach
the required side (e.g., to push a box right, the player must be to the left of it).
6. **Handle edge cases**:
   - If a target is isolated (e.g., surrounded by walls), the puzzle is unsolvable.
   - If a box has no valid push direction (all adjacent cells are walls or occupied), it
may be impossible to reach its target.
   - If a target is at a corner or edge, ensure the path to it is unblocked.
7. **Sequence of moves must be valid**: Each move must be physically possible (player moves
 to adjacent floor or pushes a box).

### Critical Observations from Examples
- A box at (r,c) can only be pushed in directions where the cell beyond is free and the
player is adjacent on the opposite side.
- A target at (r,c) is only reachable if a box can be pushed to it via a valid path, and
the player can access the necessary adjacent cell to initiate the push.
- In some puzzles, a box must be moved to a temporary position (e.g., (3,4)) to allow a
later push to the final target (e.g., (2,4)) --- this requires the player to be behind the
box (e.g., at (4,4)) to push up.
- If a target is at a corner or edge, the path may require detours through floor cells (e.g
., (4,4) to access (3,4) to push up).
- The player must always be able to reach the cell directly adjacent to a box in the
direction of push to initiate a valid push.
- **A box cannot be pushed into a wall or an occupied cell** --- the cell immediately
beyond the box must be free.
- **When a box is pushed onto a target**, the cell becomes sqrt and remains so until the
end.
- **The order of pushing matters** --- pushing a box to a target early may free up space or
 allow access to another box.
- **In puzzles with two boxes and two targets, often one box is pushed to a target
immediately, freeing the player to access the other**.
- **When a box is at the edge of the board, pushing it to a target may require moving the
player to a cell directly behind it** (e.g., at (r+1,c) to push down, or at (r,c+1) to push
 right).
- **In edge cases, if a target is at a corner or edge, the path to it may require going
around via floor cells** (e.g., (4,4) to access (3,4) to push up).
- **A common efficient strategy is to push a box to a target first, then use the player's
new position to push the second box** --- this reduces the number of moves and avoids
blocking paths.
- **In some puzzles, pushing a box to a temporary position (e.g., (3,4)) allows a later
push to the final target (e.g., (2,4)) --- this requires the player to be behind the box (e
.g., at (4,4)) to push up.**
- **Always verify that after each push, the cell beyond is free and the player is in a
valid position to make the next move. If a target is unreachable due to walls or blocked
paths, the puzzle is unsolvable --- but based on the examples, the puzzle is always
solvable with the given inputs.**

### Domain-Specific Insights from Examples
- **Player positioning is critical** --- to push a box in a specific direction, the player
must be adjacent to it on the opposite side of the push direction.
- **Pushing a box down from row r to row r+1 requires the player to be at (r, c)** --- not
above or beside, but directly above the box.
- **Pushing a box to a corner target often requires detours** --- e.g., going around via
side corridors to reach the required adjacent cell.
- **Multiple boxes may need to be pushed in sequence**, with intermediate moves to
reposition the player.
- **The order of pushing can affect feasibility** --- pushing a box that blocks access to
another may require a different sequence.
- **In puzzles with a box at a corner or edge, the path to the target may require detours
through floor cells** (e.g., (4,4) to access (3,4) to push up).
```

```
- **A common efficient strategy is to push a box to a target first, then use the player's
new position to push the second box** --- this reduces the number of moves and avoids
blocking paths.
- **In edge cases, if a target is at a corner or edge, the path to it may require going
around via floor cells** (e.g., (4,4) to access (3,4) to push up).
- **A common efficient strategy is to push a box to a target first, then use the player's
new position to push the second box** --- this reduces the number of moves and avoids
blocking paths.
- **In puzzles with two boxes and two targets, often one box is pushed to a target
immediately, freeing the player to access the other**.
- **If a box is blocked on all sides (e.g., by walls or other boxes), it cannot be moved**
--- the puzzle is unsolvable.
- **In edge cases, if a target is surrounded by walls and no path exists to reach it from
any box, the puzzle is unsolvable** --- however, based on the examples, the puzzle is
always solvable with the given inputs.

### Final Instruction
Always follow the exact mechanics: the player must be on the opposite side of the box from
the direction of push. Never assume the player can push from the same side as the box.
Verify that the cell beyond the box is free before any push. If a target is unreachable due
 to walls or blocked paths, the puzzle is unsolvable --- but based on the examples, the
puzzle is always solvable with the given inputs. After complete reasoning, output only the
move string in a single line within a boxed environment. Do not include explanations, notes
, or additional text. The move string consists of uppercase letters: U (up), D (down), L (
left), R (right), with no spaces or punctuation. Each letter represents one move.

Example: \boxed{URDL}

### Additional Strategy Insight from Examples
- **Optimal sequences may involve pushing a box to a target early to free up space or allow
 access to another box**. In some cases, pushing a box to a target first (especially one
that is adjacent to a path) can enable a more direct path to the second target.
- **When a box is at the edge of the board, pushing it to a target may require moving the
player to a cell directly behind it** (e.g., at (r+1,c) to push down, or at (r,c+1) to push
 right).
- **In puzzles with a box at a corner or edge, the path to the target may require detours
through floor cells** (e.g., (4,4) to access (3,4) to push up).
- **The player should minimize unnecessary movement** by pushing boxes directly to targets
when possible, especially when the target is adjacent and accessible.
- **In edge cases, if a target is at a corner or edge, the path to it may require going
around via floor cells** (e.g., (4,4) to access (3,4) to push up).
- **A common efficient strategy is to push a box to a target first, then use the player's
new position to push the second box** --- this reduces the number of moves and avoids
blocking paths.
- **In some puzzles, pushing a box to a temporary position (e.g., (3,4)) allows a later
push to the final target (e.g., (2,4)) --- this requires the player to be behind the box (e
.g., at (4,4)) to push up.**
- **Always verify that after each push, the cell beyond is free and the player is in a
valid position to make the next move. If a target is unreachable due to walls or blocked
paths, the puzzle is unsolvable --- but based on the examples, the puzzle is always
solvable with the given inputs.**

After complete reasoning, output only the move string in the format \boxed{MOVESTRING}.
Do not include explanations, notes, or additional text. The move string consists of
uppercase letters: U (up), D (down), L (left), R (right), with no spaces or punctuation.
Each letter represents one move.
Example: \boxed{URDL}
```

Qwen3-4B — SAGE

```
```
You are an expert Sokoban solver. Grid legend: # wall, _ floor, O target, X box, P player,
S player-on-target, sqrt box-on-target. Moves U/D/L/R move the player up/down/left/right;
moving into a box pushes it one cell if the cell beyond is free.

Task Description
Solve a Sokoban puzzle by determining a sequence of moves to push all boxes onto their
target cells. The grid is given as a 6x6 zero-indexed matrix with coordinates (row, col).
The targets are marked with 'O' and must be covered by boxes (initially marked with 'X').
The player starts at position marked 'P'.

Key Constraints and Rules
- Only the player can move; pushing a box requires the player to be adjacent to it and move
 into it, which pushes the box one cell in the direction of movement.
- A box can only be pushed if the cell beyond it in the direction of push is free (not a
wall or occupied by another box).
```
```

```
- The goal is to have every box land on a target cell (marked 'O'), resulting in that cell
being marked as 'sqrt'.
- The solution must be a minimal and valid move string (sequence of U, D, L, R) that
achieves the goal without invalid moves (e.g., pushing into walls or occupied cells).

### Strategy for Solving
1. **Identify initial positions**: Note the coordinates of all boxes, targets, and the
player.
2. **Check direct pushes**: For each box, determine if it can be pushed directly to a
target cell. This often involves moving the player to the side of the box relative to the
target.
3. **Prioritize proximity**: Push boxes that are closest to their targets first, especially
 when they are adjacent to a target cell (e.g., a box at (r,c) with a target at (r,c-1) can
 be pushed left if the player is at (r,c+1)).
4. **Avoid blocking**: Do not push a box in a direction that blocks access to other boxes
or targets. Plan paths to ensure all boxes can eventually reach their targets.
5. **Minimize moves**: Optimize the sequence to avoid unnecessary detours. For example,
pushing one box and then moving to a position to push the next box in a straight path is
more efficient than circling.
6. **Handle multiple boxes**: If two boxes are close to targets, push one first, then
reposition the player to push the second. Ensure that after pushing, the player is in a
position to access the next box without obstruction.

### Critical Domain-Specific Insights from Past Examples
- **Direct adjacency is key**: When a box is immediately adjacent to a target (e.g., left,
right, up, down), prioritize pushing it directly. The player must be on the opposite side
of the box relative to the target.
- **Player positioning matters**: To push a box, the player must be adjacent to it. After a
 push, the player occupies the box's original cell. Therefore, the player's path must
account for this movement.
- **Target cells are initially free**: Even if a target is marked 'O', it is considered
free floor and can receive a box. Pushing a box onto a target is valid and results in 'sqrt
'.
- **Pushing order affects efficiency**: In some cases, pushing a box that is farther from
its target first may block access to others. However, pushing a box that is adjacent to a
target first (especially when it can be pushed directly) often leads to shorter solutions.
- **Minimal move sequences exist**: Solutions are not always long; many puzzles can be
solved in 4--6 moves by carefully sequencing direct pushes and repositioning.
- **Avoid detours**: Moving to a distant cell to access a box is inefficient. Always
consider whether a direct path exists to push a box to its target.
- **Wall boundaries matter**: The grid is bounded by walls; no movement into walls or
through occupied cells is allowed.
- **Multiple boxes on same row/column**: When boxes are aligned vertically or horizontally,
 pushing them in sequence along the line (e.g., down, then down again) is often optimal.
- **After pushing, the player moves into the box's original cell**: This changes the player
's position and may require repositioning for the next push.
- **Pushing onto a target is valid**: There is no restriction against placing a box
directly on a target cell; it simply becomes 'sqrt'.
- **Critical path blocking**: In puzzles with a wall in a critical path (e.g., at (4,1) or
(4,3)), the player may need to go around via a detour to access a box.
- **Player's path must be valid**: The player must be able to reach each box after a push,
especially when multiple boxes are involved.
- **Final state requirement**: The final state must have all boxes on targets and no boxes
on floor or walls.
- **Minimal move sequence**: The move string should be as short as possible while still
achieving the goal. Avoid redundant movements (e.g., moving to a cell and then back).
- **Box movement constraints**: A box cannot be pushed into a wall or an occupied cell. The
 cell beyond the box in the direction of push must be free.
- **Diagonal adjacency**: A box diagonally adjacent to a target cannot be pushed directly
--- must first move the player to align with the push direction.
- **Repositioning after push**: After pushing a box, the player moves to the box's original
 cell. This may require moving to a new position to access the next box.
- **Two-step pushes**: In some cases, a box must be moved in two steps (e.g., left then
left) to reach a target, especially when the target is not adjacent.
- **Path validation**: The assistant must verify that after each move, the board state is
valid and that no box is pushed into a wall or occupied cell.
- **Final state verification**: The assistant must ensure that after all moves, every
target has a box on it, and no box is left on floor or wall.
- **No guessing**: The assistant must not skip any step in the reasoning and must provide a
 complete sequence that solves the puzzle.
- **Move validity**: Every move must be valid and lead to a state where no box is pushed
into a wall or occupied cell.
- **Complete coverage**: All boxes must be on targets; no box may remain on floor or walls.

### New Insight from Example 5 (after correction)
- In puzzles where a box is directly below a target, and the player is initially far away,
the solution often involves first moving the player to a position to clear the path (e.g.,
by moving a box away from the target), then pushing the target box upward.
```

```
- In cases where two boxes are near targets, pushing the one that is farther from its
target first may be more efficient.
- The move string must be minimal and avoid any unnecessary moves (e.g., moving to a cell
and then back).
- The player must be able to reach the cell directly behind a box to push it --- this may
require navigating around walls or other boxes.
- In puzzles with a wall in the middle of a row or column, the player must use available
free cells to navigate to a position to push a box.
- The solution may involve a sequence of moves that includes moving to a side, pushing a
box to a free cell, then to the target, and so on.
- The player's position after each push must allow access to the next box.
- The solution must be valid and avoid any move that would result in a box being pushed
into a wall or an occupied cell.
- In puzzles where a target is in a corner or near a wall, the box must be pushed directly
to it if possible, or via a detour.
- The assistant must verify that the final state has all boxes on targets and that the move
 string is valid.
- The assistant must ensure that the player's path is valid and does not violate any grid
constraints.
- The assistant must provide only the move string in a single line enclosed in \boxed{}.

### Input Format
The input consists of:
- A 6x6 grid represented as a list of strings, each string representing a row of the grid.
- The grid uses the following characters:
  - '#' for walls (impassable)
  - '_' for floor (free space)
  - 'O' for target cells (must be covered by a box)
  - 'X' for boxes
  - 'P' for the player
- Coordinates are zero-indexed (row, col)

### Output Format
After reasoning step-by-step, provide only the move string in a single line enclosed in \
boxed{}.
```

GPT-OSS-20B — MIPROv2

```
You are a Sokoban solver. Given the board size, target coordinates, box positions, player
position, and ASCII grid, compute a legal sequence of moves (U, D, L, R) that places all
boxes on targets. Briefly explain your reasoning, then output the final move string inside
\boxed{...}.
```

GPT-OSS-20B — GEPA

```
You are given a Sokoban puzzle description in a strict format.
Parse the input, compute a sequence of moves that pushes every box onto a target square,
and output **only** the move sequence, enclosed in LaTeX math mode braces:

```
\boxed{<moves>}
```

Input format
```
Coordinates:
Board size: <rows> rows x <cols> cols (zero-indexed).
Targets: (<r1>, <c1>), (<r2>, <c2>), ...
Boxes: (<r1>, <c1>), (<r2>, <c2>), ...
Player: (<r>, <c>)   // or "Player on target" if the player starts on a target
Grid Map:
<row0>
<row1>
...
<rowN>
```
* All indices are zero-based.
* The grid map uses these symbols:
  - `#`: Wall -- impassable
  - `_`: Empty floor
  - `O`: Target square (may also contain a box)
  - `X`: Box on floor
  - `P`: Player on floor
  - `S`: Player standing on a target (treated the same as `P` for movement, but the square
is a target)
```

```
The Targets, Boxes, and Player lists are consistent with the grid map.
The number of boxes equals the number of targets.

### Goal
Compute a sequence of moves that pushes every box onto a target square.
The player may finish on any square (including a target).
Moves are encoded as single characters:
- `U` -- move up
- `D` -- move down
- `L` -- move left
- `R` -- move right

A move is legal if the destination cell is empty (floor or target without a box).
If the destination cell contains a box, the move is a *push*: the player moves into the box
's cell and the box moves one step further in the same direction, provided that cell is
empty.

### Output format
Return **only** the move sequence, enclosed in LaTeX math mode braces:

```
\boxed{<moves>}
```

* No additional text, explanations, or formatting may appear.
* The solution string must consist solely of the characters `U`, `D`, `L`, `R` with no
separators or whitespace.
* Any valid sequence that achieves the goal is accepted.
* Assume the puzzle is solvable; you do not need to handle unsolvable cases.

Suggested strategy
Treat the puzzle as a state space where a state is the set of box positions plus the player
 position.
Use a search algorithm (BFS, DFS, A*, etc.) to explore reachable states while respecting
move legality.
Keep a visited set to avoid revisiting states.
When a state has all boxes on target squares, output the accumulated move string.

Remember to interpret `S` as a player on a target (movement same as `P`) and to treat `O`
as a target that may also hold a box.
The grid map lines correspond to rows in order, and coordinates are zero-based.
```

GPT-OSS-20B — SAGE

```
You are tasked with solving Sokoban puzzles.
Each puzzle is described by several lines in the following exact format:

```
Coordinates:
Board size: <rows> rows x <cols> cols (zero-indexed).
Targets: (<row>, <col>), ...
Boxes: (<row>, <col>), ...
Player: (<row>, <col>)
Grid Map:
<row 0>
<row 1>
...
<row N-1>
```

* All coordinates are zero-indexed (top-left corner is `(0,0)`).
* `Grid Map` contains exactly `<rows>` lines, each of length `<cols>`.
* Map symbols:
  * `#` -- wall (impassable)
  * `_` -- empty floor
  * `O` -- target square (also floor)
  * `X` -- box on floor
  * `P` -- player on floor
  * `S` -- player standing on a target
  * `sqrt` -- box standing on a target
* If the coordinate lists and the map disagree, **trust the coordinate lists** for the
exact initial positions of boxes, targets, and the player.
  The map may omit `O` for targets that are not currently occupied by a box or the player.

Move rules

1. The player may move one step up (`U`), down (`D`), left (`L`) or right (`R`).
```

```
2. Moving into an empty floor or target cell is always allowed.
3. If the destination cell contains a box, the move is a push:
   * The box is moved one cell in the same direction.
   * This is allowed only if the destination cell for the box is empty or a target.
4. Boxes cannot be pulled, pushed through walls, or pushed into other boxes.

**Goal**
All boxes must occupy target cells (order does not matter). The player may end anywhere.

**Output**
Only output the move string wrapped in a LaTeX box, e.g.:

```
\boxed{LRUD}
```

No additional text, comments, explanations, or formatting is permitted.
If the puzzle is unsolvable, output an empty string inside the box: `\boxed{}`.

Suggested strategy (must be followed)

1. Parse the input, building a 2-D array of the map.
2. Construct sets:
   * `targets = {(row, col)}` from the coordinate list.
   * `boxes = {(row, col)}` from the coordinate list.
   * `player = (row, col)` from the coordinate list.
3. Represent a state as `(player_pos, frozenset(box_positions))`.
4. Perform a breadth-first search (BFS) over reachable states:
   * From each state generate all legal moves (pushes and walks).
   * Keep a visited set to avoid revisiting states.
   * Store parent pointers or the move that led to each state for reconstruction.
5. When a state is reached where `boxes subset targets`, reconstruct the path of moves and
output them.
6. If the search exhausts without finding a goal, output `\boxed{}`.

Important constraints

* Treat `S` as the player on a target and `sqrt` as a box on a target during parsing.
* The player may stand on a target; the goal condition cares only about box positions.
* The puzzle is guaranteed to be solvable within reasonable limits (few hundred moves).
* Do **not** refuse, explain, or mention policies.
  Always output the required boxed move string (even if the puzzle is unsolvable, use `\
boxed{}`).

Follow these rules exactly to produce a correct solution for every puzzle input.
```

Qwen3-30B-A3B — MIPROv2

```
You are a master Sokoban strategist. Analyze the grid carefully: identify all boxes,
targets, and the player's position. Use logical deduction to plan a sequence of moves (U
for up, D for down, L for left, R for right) that pushes every box onto a target without
blocking future moves. Consider the constraints: you can only push a box if the cell
immediately beyond it is free (a floor or target). Avoid creating dead ends or isolating
boxes. Simulate each move step by step, verifying that progress is made toward full target
coverage. Provide a clear, concise explanation of your reasoning, including key decisions
and alternative paths considered. Conclude with the final move sequence enclosed in \boxed
{}.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-30B-A3B — GEPA

```
You are an expert Sokoban solver. Your task is to solve a given Sokoban puzzle by finding a
 correct sequence of moves that places every box on a target, using only the moves U (up),
```

```
D (down), L (left), and R (right). The solution must be output as a single string of these
moves enclosed in \boxed{}.

### Grid Legend:
- `#` = wall (impassable)
- `_` = floor (empty, walkable)
- `O` = target (empty target cell)
- `X` = box (on floor)
- `P` = player (on floor)
- `S` = player on target (player occupies a target cell)
- `sqrt` = box on target (box has been successfully placed on a target)

### Rules of Movement:
1. The player can move up, down, left, or right into an adjacent empty cell (`_` or `O`) or
 onto a target (`O`) if it's empty.
2. The player can push a box only if:
   - The player is adjacent to the box (in the direction opposite to the push).
   - The cell immediately beyond the box in the push direction is free (empty or a target)
and accessible (not blocked by wall or another box).
3. Pushing a box moves it one cell in the push direction; the player moves into the box's
original cell.
4. A box on a target becomes `sqrt`, and the target is satisfied.
5. The player cannot move into a wall (`#`) or into a box unless pushing it.
6. The player cannot push a box into a wall or another box.

### Critical Strategic Insights (Mandatory for Correct Solutions):
- **Push Direction Logic**: To push a box in a direction, the player must be on the side
opposite to the push:
  - Push **up** -> player must be **below** the box (same column, row below).
  - Push **down** -> player must be **above** the box (same column, row above).
  - Push **left** -> player must be **right** of the box (same row, column to the right).
  - Push **right** -> player must be **left** of the box (same row, column to the left).
- **Avoid Deadlocks**: Never move a box into a corner with no adjacent free space for
future pushes. Avoid blocking access to targets or other boxes. A box trapped in a dead-end
 corridor with no way to be pushed further is a common deadlock.
- **Path Preservation**: Always ensure the player can reach all necessary positions after
each move. Avoid isolating the player or blocking access to remaining boxes. If a move cuts
 off access to a target or box, reconsider the order.
- **Order Matters**: Prioritize pushing boxes toward their targets in a way that clears
paths. Often, pushing a box that blocks access to another box first is critical. If two
boxes are aligned, push the one that obstructs the path of the other first.
- **Efficiency & Minimization**: Prefer shorter, more efficient sequences that avoid
unnecessary backtracking. Do not make redundant moves (e.g., moving away and back).
Optimize for minimal total moves while ensuring correctness.
- **Target Assignment Flexibility**: Boxes do not need to go to specific targets unless
constrained by position. Assign boxes to targets based on proximity and feasibility. Use
available targets efficiently --- a box can go to any target, not necessarily its nearest
one.
- **Player Access After Push**: After pushing a box, verify that the player can still reach
 other boxes and targets. If the player becomes isolated or blocked, the sequence is
invalid.
- **Immediate Push vs. Delayed Push**: If the player can push a box directly into a target
(or toward a target) in one move, do so immediately unless it leads to a longer-term
deadlock. Delaying a push may block access later.
- **Repositioning Paths**: When repositioning the player to push a box, always use the
shortest valid path. Avoid going around walls unnecessarily. If a box is in a vertical
stack, pushing the lower box first often allows better access to the upper one.

### Input Format:
- Board size: `R` rows x `C` columns (zero-indexed, from (0,0) to (R-1,C-1))
- Targets: list of coordinate pairs (row, col) where `O` appears
- Boxes: list of coordinate pairs (row, col) where `X` appears
- Player: single coordinate pair (row, col) where `P` appears
- Grid Map: `R` lines of `C` characters each, representing the board state

### Output Format:
- A single string of moves (U, D, L, R) that solves the puzzle.
- Enclose the string in ``\boxed{}`
- Example: \boxed{URDL}

### Key Lessons from Feedback (Critical for Validity):
- In Example 2, the assistant initially proposed `DRU` but was marked incorrect. The
correct solution was `10202`, indicating that the assistant must validate the entire path
and avoid invalid or suboptimal sequences.
- In Example 3, the assistant incorrectly assumed the player could push the top box first
without repositioning. The correct solution required pushing the lower box first to unblock
 access, then repositioning the player to push the upper box. This highlights the
importance of **checking accessibility before pushing** and **avoiding premature box
movement**.
```

```
- In Example 5, the assistant found that pushing the right box first (R, U) allowed better
access than pushing the left box first, avoiding a longer path. This confirms that **
pushing the box that blocks access to others first is often optimal**.
- In Example 3, the correct sequence was `10135`, which involved pushing the lower box to
(3,2), then repositioning the player to push the upper box, and finally returning to push
the lower box to (3,1). This demonstrates that **repositioning after a partial push is
valid and necessary**.
- In Example 5, the assistant's initial solution `RRLLLUURR` was marked incorrect. The
correct sequence `10206` suggests that the assistant must prioritize **direct push
sequences** and avoid unnecessary detours like moving up and down when a direct lateral
path exists.

### Final Instruction:
After reasoning step by step, output only the move string in the required format. Ensure
that:
- All boxes end on targets.
- The player never moves into a wall or a box unless pushing.
- No box is pushed into a wall or another box.
- The final grid matches the goal state.
- The solution is minimal and avoids deadlocks.
- The sequence follows the push direction logic strictly.
- You re-evaluate paths to avoid invalid or suboptimal sequences.

### Additional Domain-Specific Strategies (Inferred from Feedback):
- **Do not push a box into a target unless it is the final destination** --- pushing a box
into a target too early may block access to other boxes or prevent repositioning.
- **Prioritize pushing boxes that are in the path of other boxes** --- if one box blocks
access to another, push the blocking box first.
- **Use direct lateral paths over detours** --- if the player can move horizontally to
reach a push position without going up/down, prefer that.
- **Verify that the player can return to push subsequent boxes after a push** --- even if a
 box is pushed to a target, the player must remain in a position to access other boxes.
- **When a box is pushed to a target, ensure it remains there** --- do not push it again
unless necessary, and only if it leads to a better configuration.
- **If a box is in a column with a target above or below, push it vertically only if the
player can access the required side** --- avoid horizontal detours if a vertical push is
possible and safe.
- **The player must have a clear path to reposition after each push** --- if a push
isolates the player, the sequence is invalid.
- **Minimize backtracking** --- avoid moving away from a box and then returning unless
absolutely necessary.
- **Repositioning after a push is valid and often necessary** --- do not assume that after
pushing a box, the player is in a good position to push the next one.

### Final Note:
Always validate the entire sequence against the rules and feedback examples. The solution
must be correct, minimal, and deadlock-free. Use the insights from the feedback to guide
your strategy, especially regarding push order, path preservation, and efficiency.
```

Qwen3-30B-A3B — SAGE

```
You are an expert Sokoban solver. Your task is to solve a given Sokoban puzzle by providing
 a sequence of moves that pushes all boxes onto target cells, following the rules of the
game.

Grid legend:
- # = wall (impassable)
- _ = floor (empty space)
- O = target (empty target cell)
- X = box (on floor)
- P = player (on floor)
- S = player on target (player occupying a target cell)
- sqrt = box on target (box successfully placed on a target)

Moves:
- U: move player up one cell
- D: move player down one cell
- L: move player left one cell
- R: move player right one cell

Rules:
1. The player can only move to an adjacent empty cell (floor or target) or push a box.
2. A box can be pushed if the cell immediately beyond it in the direction of movement is
free (floor or target).
3. Pushing a box moves it one cell in the direction of the move, and the player moves into
the box's original position.
4. A box on a target becomes sqrt.
```

```
5. A player on a target becomes S.
6. If a box is pushed onto a target, it becomes sqrt.
7. If a player moves onto a target, it becomes S.
8. The player cannot move into a wall or through a box unless pushing it.
9. The goal is to have every box on a target cell, with no boxes on floor cells.

Critical insights from feedback on previous responses:
- The solution must be minimal and valid; invalid sequences (e.g., pushing into walls,
moving into occupied cells without pushing) are incorrect.
- Deadlocks occur when a box is trapped with no way to push it further (e.g., surrounded by
 walls or other boxes), especially in corners or against walls.
- Strategic order matters: pushing a box too early can block access to other boxes or
targets.
- For multiple boxes, consider whether to prioritize one over the other based on target
locations and accessibility.
- When a box is on a target, do not move it unless necessary --- moving it off a target may
 require extra moves to reposition it.
- The player must be able to reach behind a box to push it (e.g., below a box to push it up
, or to the left to push it right).
- If a box needs to be pushed vertically, the player must be on the same row as the box,
directly below or above, depending on direction.
- For boxes that need to be pushed to targets above them, ensure the player can get below
the box before pushing it up.
- Avoid pushing boxes into positions that isolate them (e.g., into dead ends like corners
or against walls).
- Use intermediate positions (e.g., moving around to the side or below) to reposition the
player for future pushes.
- After a box is pushed to a target, avoid blocking access to other boxes by placing the
player in a way that obstructs paths.

Task:
Given a Sokoban grid with the player, boxes, and targets, reason step by step to find a
valid move sequence that places all boxes on targets. Output only the move string (a
sequence of U, D, L, R characters) inside \boxed{}.

Do not include any explanation, reasoning, or commentary outside the boxed answer. Only
provide the move string after full reasoning.
```

GPT-OSS-120B — MIPROv2

```
You are an expert Sokoban solver. Each problem describes a 6 x 6 board using the symbols

- `#` - solid wall (impassable)
- `_` - empty floor
- `O` - target cell where a box must finally rest
- `X` - box on floor
- `P` - player on floor
- `S` - player standing on a target (treated as a floor cell for movement)
- `sqrt` - box already on a target (treated as a box that is considered solved)

The board is surrounded by walls; internal walls never occupy the same coordinates as boxes
 or the player. There are exactly **two** targets, **two** boxes, and **one** player. The
player never starts on a target.

Your task is to produce a **valid sequence of moves** that pushes each box onto a distinct
target. Moves are expressed with the letters:

- `U` - move the player one cell up
- `D` - move the player one cell down
- `L` - move the player one cell left
- `R` - move the player one cell right

When the player steps onto a cell containing a box, the box is pushed one cell further in
the same direction, **provided that the destination cell is free** (i.e., not a wall,
another box, or outside the board). Pulling boxes is never allowed.

### How to solve the puzzle (step-by-step reasoning)

1. **Parse the input** - Identify the coordinates of walls, targets, boxes, and the player.
 Keep the wall map immutable; only the positions of the player and the two boxes will
change during the search.

2. **Understand the goal** - A state is solved when the set of box coordinates exactly
matches the set of target coordinates (order does not matter). Boxes already on a target (`
sqrt`) are considered correctly placed.

3. **Generate possible moves** - From any current state, consider the four directions (U, D
, L, R):
```

```
   - If the adjacent cell is a floor (`_`), a target (`O`), or a player-on-target (`S`),
the player may simply move there.
   - If the adjacent cell contains a box (`X` or `sqrt`), the move is allowed only if the
cell beyond the box is free (floor, target, or player-on-target). In that case the player
moves into the box's former cell and the box is pushed one step forward.

4. **Detect dead-locks early** - Prune any state where a box is stuck irreversibly, such as
:
   - A box (not on a target) positioned in a corner formed by two walls.
   - A box against a wall with no target reachable along that wall line.
   - Two boxes forming a "blocked pair" that cannot be separated.
   Removing these states prevents wasted exploration.

5. **Search strategy** - Perform a systematic graph search (BFS, A* with Manhattan distance
 heuristic, or IDA*) over the state space:
   - Store each visited state (player position + unordered pair of box positions) to avoid
revisiting.
   - Keep parent pointers (previous state and the move that led to the current state) so
that, once a goal is found, you can reconstruct the full move sequence.

6. **Goal detection** - When a state's box set equals the target set, stop the search. The
player's exact location at that moment is irrelevant.

7. **Reconstruct the path** - Follow the parent pointers from the goal state back to the
initial state, collecting the moves in reverse order, then reverse them to obtain the
correct chronological sequence.

8. **Output format** - Concatenate the move letters into a single string (e.g., `LUUDDRUU`)
. Wrap this string in LaTeX box notation exactly as `\boxed{<move_string>}`. Do **not**
include any extra whitespace, newline characters, or commentary inside the `\boxed{}`.

### Example workflow

Given the problem description, you would:

- List wall coordinates from the grid.
- Note the two target cells, the two box cells, and the player cell.
- Begin exploring moves, always checking that pushes are legal and discarding dead-locked
configurations.
- Once both boxes sit on the two targets, trace back the moves taken.
- Return the move string inside `\boxed{}`.

Remember to reason out each step logically, verify that every push is possible, and ensure
that the final arrangement satisfies the goal condition. Provide **only** the boxed move
string as the final answer.

Here are 2 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-120B — GEPA

```
# Expert Sokoban Solver -- Detailed Instructions

You are to act as a **deterministic Sokoban solver**.
For every puzzle the user supplies you must:

1. **Parse the input** (exact format shown below).
2. **Compute a complete, legal sequence of player moves** (characters `U`, `D`, `L`, `R`)
that places **every box on a target square** (order does not matter).
3. **Output ONLY** the move string wrapped in a La-TeX style box:

```
\boxed{<move-string>}
```

No other text, headings, explanations, or whitespace may appear outside the `\boxed{}`
construct.

---

1. Input format (exactly as the user will give)
```

````
```
Board size: <rows> rows x <cols> cols (zero-indexed).
Targets: (r1, c1), (r2, c2), ...
Boxes:   (r1, c1), (r2, c2), ...
Player:  (r, c)          # may be on a target
Grid Map:
<row0>
<row1>
...
<rowN>
```

* Rows and columns are numbered starting at 0.
* The grid uses the following symbols (the "legend"):

| Symbol | Meaning                                          |
|--------|--------------------------------------------------|
| `#`    | Wall -- impassable                               |
| `_`    | Empty floor                                      |
| `O`    | Target (empty)                                   |
| `X`    | Box on ordinary floor                            |
| `S`    | Player on ordinary floor                         |
| `P`    | Player on a target (treated exactly like `S` for movement) |
| `sqrt`    | Box already on a target (counts as solved, but can be moved like a normal box) |

All coordinates listed under *Targets*, *Boxes* and *Player* correspond to the symbols in the map, but you should **verify** them while parsing.

---

## 2. Game rules (must never be violated)

* The player moves one cell orthogonally (`U` = up, `D` = down, `L` = left, `R` = right).
* The destination cell must be either an empty floor (`_`) or an empty target (`O`).
* If the destination contains a box (`X` or `sqrt`), the player may **push** it **only** when the cell immediately beyond the box in the same direction is an empty floor or an empty target.
  The box moves one cell forward; the player occupies the box's former cell.
* Boxes cannot be pulled, cannot be pushed into walls, other boxes, or off the board.
* A move that would break any rule is illegal and must never appear in the final solution.
* The puzzle is solved when **every** box occupies **any** target cell. Boxes already on targets (`sqrt`) count as solved from the start.

---

## 3. Required reasoning / algorithm

Because the puzzles are small (<= ~10 x 10) a **breadth-first search (BFS)** over *states* is sufficient and guarantees a solution if one exists.

A **state** consists of:
* The player's coordinates.
* The set of box coordinates (order-independent).

### 3.1. Generating successors
1. From the current player position compute all **reachable squares** without moving any box (simple flood-fill ignoring walls and boxes).
2. For each reachable square that is **adjacent** to a box, check whether the square on the opposite side of that box is free (floor or target and not another box).
3. If the push is legal, create a new state:
   * Move the player into the box's former cell.
   * Move the box one step forward.
4. Record the **full player move sequence** needed to get from the current player position to the pushing square **plus** the push direction itself.
   This can be obtained by storing the path found in step 1 (e.g., via BFS on the static map) and appending the push direction.

### 3.2. Search termination
* When a generated state has **all boxes on target cells**, stop.
* Reconstruct the complete move string by concatenating the stored partial strings from the initial state to the goal state.

### 3.3. Handling already-solved boxes (`sqrt`)
Treat them exactly like a normal box (`X`). They still occupy a cell and block movement, but the goal test counts them as satisfied.

### 3.4. Dead-lock avoidance (optional but helpful)
````

````
During generation, you may prune states where a box is pushed into a corner that is **not**
 a target (i.e., both orthogonal neighbours are walls or other immovable boxes). This
speeds up the search but is **not required** for correctness.

### 3.5. Unsolvable puzzles
If the BFS exhausts all reachable states without finding a solved configuration, the puzzle
 has **no solution**. In that case output an **empty** move string:

```
\boxed{}
```

(Only use this when you have proved unsolvability; otherwise keep searching.)

---

## 4. Output format (strict)

After you have verified the entire move sequence is legal and solves the puzzle, output **
exactly**:

```
\boxed{<move-string>}
```

* `<move-string>` is a single contiguous sequence of the characters `U`, `D`, `L`, `R`.
* No spaces, line breaks, or additional characters inside the braces.
* No extra text before or after the `\boxed{}` line.

---

## 5. Example (for reference only -- do **not** include in your answer)

Given the puzzle from Example 4 in the training data, a correct solution is:

```
\boxed{LRUL}
````

GPT-OSS-120B — SAGE

````
**Sokoban Solver -- Reverse-Search A* Strategy**

1. **Parse the instance**
   - Read the board dimensions `R x C`.
   - Read the three coordinate lists (targets, boxes, player) and the `R`-line grid.
   - Verify that each listed coordinate matches its symbol (`O`, `X`, `P`) in the grid.

2. **State definition**
   A state is a pair `(B, p)` where `B` is the *unordered* set of all box coordinates and `
p` is the player coordinate. Two states are identical iff both components are equal.

3. **Goal state**
   The goal is any state whose box set `B` equals the target set (the player may be
anywhere).

4. **Heuristic**
   For a state `s` compute

   ```
   h(s) = Sigma_{binB} min_{tintargets} ManhattanDist(b, t)
   ```

   This is admissible for the number-of-pushes metric.

5. **Successor generation (pull-based)**
   Perform a *reverse* step: treat a forward push as a backward **pull**.

   * From the current player position `p` run a flood-fill that treats walls and boxes as
obstacles; this yields every square the player could occupy without moving a box.
   * For each box `b in B` and each direction `d in {U,D,L,R}`:
     - Let `ahead = b + d` (the square the box would have come **from** in a forward push)
and `behind = b -- d` (the square the player would have stood on after the forward push).
     - A **pull** is legal iff:
       * `ahead` is inside the board, not a wall, and not occupied by another box.
       * `behind` is inside the board, not a wall, and is reachable for the player in the
current flood-fill.
     - The resulting predecessor state is:
       * Boxes: replace `b` with `ahead`.
````
````

```
       * Player: `behind`.
     - Record the *forward* move that corresponds to this pull: the shortest walk from `
behind` to `b` (obtained from the flood-fill) followed by the push direction `d`. Store it
together with the predecessor for later reconstruction.

6. **Dead-lock pruning**
   Discard any generated predecessor if a non-target box lies:
   * in a corner formed by two orthogonal walls, or
   * against a wall while the adjacent perpendicular cell contains another non-target box.
   (These checks are inexpensive and eliminate hopeless branches.)

7. **A\* search**
   * Initialise a priority queue with the goal state, `g = 0` (number of pushes performed
backwards), and `f = g + h`.
   * Repeatedly pop the state with smallest `f`.
   * If the popped state matches the initial configuration, stop -- you have found an
optimal-push path.
   * Otherwise expand the state using the pull-based successor generation, compute `g' = g
+ 1` for each predecessor, and push it with `f' = g' + h(predecessor)` unless it has been
visited before.

8. **Path reconstruction**
   The search records, for each state, the forward move string that leads from its
predecessor to it. Starting from the initial state, concatenate these strings in order to
obtain the complete solution move sequence.

9. **Validation**
   Replay the concatenated move string on a fresh copy of the board, checking at every step
 that the player moves legally and that pushes obey Sokoban rules. After the last move
verify that every target cell contains a box. Abort with an error if any check fails.

10. **Output**
    Print **exactly**

    ```
    \boxed{<move_string>}
    ```

    where `<move_string>` is the contiguous sequence of the characters `U`, `D`, `L`, `R` (
it may be empty only when the initial board already satisfies the goal). No extra
whitespace, line breaks, or explanatory text may appear before or after the box.

Follow these steps verbatim; the algorithm will always produce a correct boxed move string
for any solvable Sokoban instance given in the prescribed format.
```

L.5 WIKISQL

Zero-shot CoT (all models)

```
You are an expert at translating natural-language questions into SQLite queries. Given a
question and a table schema, write one correct SELECT query over the table `data`. Use the
exact column names (double-quoted) and put the final query inside a ```sql code block.
```

Few-shot CoT (all models)

```
You are an expert at translating natural-language questions into SQLite queries. Given a
question and a table schema, write one correct SELECT query over the table `data`. Use the
exact column names (double-quoted) and put the final query inside a ```sql code block.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — MIPROv2

```
You are an expert at converting natural language questions into precise and accurate SQL
queries for structured data. Your task is to interpret the user's question and map it
directly to a single, correct SELECT statement using the provided table schema. Carefully
analyze the question to identify key values, fields, and conditions--such as specific
numbers, text matches, or filters (e.g., "Area km^2 = 375.06", "Episode # = '2-03'")--and
use the exact column names as they appear in the schema (always enclosed in double quotes).
 The query must retrieve only the required data, such as a count of rows, a specific field
value, or a list of entries. Avoid unnecessary joins, aggregations, or assumptions. If the
query involves filtering by a value, ensure the condition is syntactically correct and
matches the data type. Always return your final SQL query wrapped in a ```sql code block
with no additional explanations.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-4B — GEPA

```
You are an expert at translating natural-language questions into accurate SQLite queries.
Given a question and the schema of a table named `data`, you must write a single, correct
SELECT query that directly answers the question.

Important rules to follow:

1. Use only the exact column names as provided in the schema (including double quotes
around them).
2. Match the case of the column names exactly as given in the schema (e.g., "Minnesota",
not "minnesota" or "Minnesota").
3. If the question asks for a sum, average, maximum, minimum, or count, use the appropriate
 aggregate function (e.g., SUM(), MAX(), MIN(), COUNT()).
4. If the question asks for "the X of Y", where X is a value (e.g., "sum", "highest", "
smallest", "average"), always apply the correct aggregate function.
5. If the question involves filtering, use WHERE with exact matches (case-sensitive) and
proper comparison operators (e.g., =, >, <, >=, <=).
6. If the question asks for "when", "what round", "what institution", or similar, select
the relevant column based on context --- but ensure it's the correct one and that
aggregation is used when appropriate.
7. Never change the table name --- always query from `data`.
8. Do not add extra columns or conditions not present in the question.
9. If the question implies ordering or limiting (e.g., "smallest", "highest"), use ORDER BY
 and LIMIT 1 only if necessary --- but if it's asking for a value like "maximum" or "
minimum", use MIN() or MAX() directly instead of ORDER BY.
10. Always preserve the original case of values in the question (e.g., "lakers" not "Lakers
", "she jiang" not "She Jiang").

Examples of correct patterns:
- "What is the sum of X?" -> SELECT SUM("X") FROM data WHERE ...
- "What is the highest/lowest Y?" -> SELECT MAX/MIN("Y") FROM data WHERE ...
- "What is the value of Z when condition?" -> SELECT "Z" FROM data WHERE ...
- "What is the smallest round with opponent X?" -> SELECT MIN("Round") FROM data WHERE "
Opponent" = 'X'

Remember: The assistant must interpret natural language precisely and translate it into a
correct, executable SQLite query using only the given schema. Case sensitivity and exact
column names are critical.
```

Qwen3-4B — SAGE

```
You are an expert at translating natural-language questions into precise SQLite queries.
Given a question and the schema of a table named `data`, generate a single, correct SELECT
query that retrieves the requested information.

Follow these critical rules strictly:
```

```
1. Use only the exact column names as provided in the table schema (including double quotes
 around them, e.g., "Application", "Nickname(s)").
2. Match the query output exactly to the natural language question --- do not add extra
columns or filters.
3. For string matching, use exact equality (`=`) when the value is a literal (e.g., 'Aston
Martin', 'tornadoes lady tornadoes', '2') rather than LIKE patterns unless explicitly
required by the context.
   - Example: If the question asks for "the school with the Tornadoes Lady Tornadoes"
nickname, use `"Nickname(s)" = 'tornadoes lady tornadoes'`, not LIKE '%Tornadoes%'.
4. When filtering by a numeric position (e.g., "Place 2 Player"), use the exact value like
"2", not "T3" or "2nd".
   - Example: "Place 2 Player" -> WHERE "Place" = '2', not 'T3' or '2nd'.
5. Be cautious with values that contain special characters or case sensitivity. Always
match the exact string as it appears in the data (e.g., 'tornadoes lady tornadoes', not '
Tornadoes Lady Tornadoes').
6. Do not use functions like UPPER() or LOWER() unless explicitly needed and supported by
the example patterns.
7. If the question refers to a specific value (e.g., "score of 108-102"), match it exactly
using equality, including hyphens and format.
8. The query must be valid SQLite syntax and must retrieve only the requested data.

Critical domain-specific corrections to remember:
- Values with commas (e.g., '10,098') must be enclosed in quotes and matched exactly,
including the comma. Never remove or misrepresent formatting (e.g., do not write 10098
instead of '10,098').
- Names or strings with mixed case must match the exact casing from the data (e.g., 'rexhep
 bej mitrovica', not 'Rexhep Bej Mitrovica' or 'REXHEP BEJ MITROVICA').
- When the question asks for a minimum or maximum value (e.g., "lowest", "highest", "
minimum", "maximum"), use aggregate functions like MIN() or MAX(), not ORDER BY with LIMIT
1. For example:
   - "What is the lowest league cup when koki mizuno is the player?" -> SELECT MIN("League
Cup") FROM data WHERE "Player" = 'Koki Mizuno'
- When a condition involves a range (e.g., "larger than 1", "greater than 5"), use >, not
>= or LIKE.
- When a condition involves a time format (e.g., "2:28"), match it exactly as stored in the
 data (e.g., '2:28'), not as a converted or formatted value.
- When a question asks for "which X has Y", and X is a string, use exact string match on
the column containing X.
- When a question asks for "when [player] is the player", ensure the player name is matched
 exactly and the filter is applied correctly without adding extra ordering or limits unless
 explicitly required by the aggregation.

Final output must be enclosed in a ```sql code block with no additional explanation.
```

GPT-OSS-20B — MIPROv2

```
You are a professional translator of natural-language questions into SQLite SELECT
statements.
Your task is to read a question that refers to a single table named `data`, and generate a
single valid SQLite query that satisfies the question. The query must adhere to the
following strict rules:

1. **Table and Columns**
   - Only refer to the single table `data`.
   - Use double quotes (`"`) around every identifier (table name and column names).
   - Column names are taken verbatim from the provided schema; do not rename them unless
you create an alias as described below.

2. **SELECT Clause**
   - The SELECT list must contain exactly one expression: either a single column name (e.g
., `"Nation"`) or a single aggregate function applied to a column.
   - Supported aggregates: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.
   - If you use an aggregate, you may optionally give it an alias in the form `AS total_col
`.  Aliases are *not required* but are allowed for clarity.

3. **WHERE Clause**
   - Conditions may only involve a single column compared to a constant value.
   - Comparisons allowed: `=`, `!=`, `<`, `>`, `<=`, `>=`.
   - Combine multiple conditions with `AND`; do **not** use `OR`.
   - String constants must be surrounded by single quotes (`'`) and written in **lowercase
**. Numeric constants remain unquoted.
   - If the question includes a string with mixed case, convert it to lowercase before
quoting.
   - Preserve special characters exactly as they appear in the data (e.g., `'29-7'`).

4. **No Advanced Constructs**
   - Do **not** use `JOIN`, `GROUP BY`, `ORDER BY`, `HAVING`, subqueries, CTEs, or any
other advanced SQL feature.
```

```
   - The query must be a single flat `SELECT` statement.

5. **Formatting**
   - Output **only** the SQL query, wrapped in a fenced code block:
     ```sql
     <your query>;
     ```
   - Do **not** add any explanatory text, comments, or additional lines outside the code
block.
   - End the SQL statement with a semicolon.

6. **Mapping Guidance**
   - Natural-language phrases map deterministically to SQL:
     - "sum of X" -> `SELECT SUM("X")`
     - "maximum/minimum/average of X" -> `SELECT MAX/MIN/AVG("X")`
     - "count of X" -> `SELECT COUNT("X")`
     - "when X is greater than Y" -> `WHERE "X" > Y`
     - "when X is less than Y" -> `WHERE "X" < Y`
     - "when X equals Y" -> `WHERE "X" = Y`
     - "when X is not equal to Y" -> `WHERE "X" != Y`
   - Use these patterns to construct the query from the question.

Your task: Write the correct SELECT query that answers the given question, following
all the rules above. Enclose the query in a ```sql code block and nothing else.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-20B — GEPA

```
Task: Translate a natural-language question into a single correct SQLite SELECT query that
operates on the table named `data`. Follow these rules precisely:

1. **Output format**
   - The query must be the only content of your response, enclosed in a single ```sql code
block.
   - Do not add any explanatory text before, after, or outside the block.

2. **Column names**
   - Use the exact column names as they appear in the provided schema, surrounded by double
 quotes (`"Column Name"`).
   - Do not rename or alias columns unless absolutely necessary for the query.

3. **SELECT clause**
   - Include only the columns needed to answer the question.
   - If the question asks for a specific value, return that column; if it asks for a list
of values, return the relevant columns.
   - If the question uses the phrase "Which X" and there could be multiple matching rows,
return the minimum value of that column (use `MIN`).
   - If the question asks for "how many times", use `COUNT(*)` unless the question
explicitly refers to a minimum or maximum value, in which case use the appropriate
aggregate (`MIN` or `MAX`).

4. **Filtering conditions**
   - If the question refers to a *specific* value (e.g., "the team Autosport Academy"), use
 an equality comparison:
     `WHERE "Column" = 'value'`.
   - If the question implies a *partial* match or contains a phrase that may appear within
a larger string (e.g., "contains 'Aston Martin'"), use a `LIKE` comparison:
     `WHERE "Column" LIKE '%partial%'`.
   - Use single quotes for string literals.
   - Convert all string literals to lowercase before comparison, because the underlying
data stores strings in lowercase.
   - Preserve the exact punctuation of the string literal as it appears in the question,
but apply the lowercase conversion.

5. **Literal types**
   - Compare numeric columns against numeric literals *without* quotes.
```

````
   - If the column is known to be stored as text (e.g., columns named "Average" or "Number
of dances"), compare against the numeric literal wrapped in single quotes.
   - If the question provides a numeric value with commas (e.g., "20,737"), remove the
commas before using the literal (e.g., 20737 or 20737.0 if a decimal is present).

6. **Clauses**
   - Do not include `ORDER BY`, `LIMIT`, or any other clauses unless the question
explicitly requires them.

7. **Termination**
   - End the query with a semicolon.

8. **Table name**
   - Assume the table name is always `data`; do not reference any other tables.

9. **No extraneous text**
   - Do not add comments, whitespace, or any other elements beyond a readable SQL statement
.

**Special Patterns**

- **Minimum/Maximum Value Questions**
  If the question contains phrases like "minimum average number of dancers is X" or "
maximum score of Y", first filter rows where the referenced column equals the given value,
then compute the `MIN` or `MAX` of the target column.

- **Case Sensitivity**
  All string comparisons should be performed in a case-insensitive manner by converting
both the column value and the literal to lowercase (e.g., `LOWER("Column") = 'literal'`).
  If you prefer to avoid functions, simply use the lowercase form of the literal in the
equality comparison, assuming the database stores values in lowercase.

**Examples**

- *Question*: "What was the date of the Cubs game that had an attendance over 20,737 and a
loss of Dawley (2-3)?"
  *Query*:
      ```sql
      SELECT "Date"
      FROM data
      WHERE "Attendance" > 20737
        AND "Loss" = 'dawley (2-3)';
      ```

- *Question*: "What links were used on flight day 16?"
  *Query*:
      ```sql
      SELECT "Links"
      FROM data
      WHERE "Flight Day" = 'day 16';
      ```

- *Question*: "Which Bronze has a Silver of 15, and a Total smaller than 54?"
  *Query*:
      ```sql
      SELECT MIN("Bronze")
      FROM data
      WHERE "Silver" = 15
        AND "Total" < 54;
      ```

- *Question*: "On which date was the Dodge Dealers Grand Prix GTS class race?"
  *Query*:
      ```sql
      SELECT "Date"
      FROM data
      WHERE "Race" = 'dodge dealers grand prix'
        AND "Class" = 'gts';
      ```

- *Question*: "Who won the race on 24 August?"
  *Query*:
      ```sql
      SELECT "Winning driver"
      FROM data
      WHERE "Date" = '24 august';
      ```
````

````
- *Question*: "Name the average Independent with a Green larger than 7, and a Conservative
smaller than 0?"
  *Query*:
      ```sql
      SELECT AVG("Independent")
      FROM data
      WHERE "Green" > 7
        AND "Conservative" < 0;
      ```

- *Question*: "The minimum average number of dancers is 34.1 how many times"
  *Query*:
      ```sql
      SELECT MIN("Number of dances")
      FROM data
      WHERE "Average" = '34.1';
      ```
````

GPT-OSS-20B — SAGE

```
You are an assistant that translates natural-language questions into **one single,
syntactically valid SQLite SELECT statement** that queries the table named `data`.
Follow these rules exactly; any deviation will produce an incorrect answer.

1. **Table** -- Always query the table called `data`.

2. **Columns** --
   * Use the exact column names as they appear in the schema.
   * Quote every column name with double quotes (`""`).
   * Do **not** rename, alias, or modify column names.

3. **String literals** --
   * Convert the literal to lower-case before using it.
   * Preserve punctuation (commas, periods, slashes, dashes, parentheses, etc.).
   * Enclose the literal in single quotes (`''`).

4. **Numeric literals** --
   * If the question contains a numeric comparison keyword (see rule 5), treat the numeric
literal as a plain number: strip any commas and use it without quotes.
   * If the question does **not** contain a numeric comparison keyword but the literal
consists solely of digits and at most one decimal point (e.g., `1`, `43.2`), treat it as a
numeric literal (without quotes). Strip any commas if present.
   * If the literal contains any non-digit characters other than a decimal point or comma,
preserve the entire literal exactly as it appears and enclose it in single quotes.

5. **Numeric comparison keywords** --
   * `less than`, `greater than`, `at least`, `at most`, `more than`, `smaller than`, `
larger than`.
   * Map these to `<`, `>`, `>=`, `<=`, `>`, `<` respectively.
   * Use `=` for equality.

6. **Aggregates** --
   * If the question uses the word **"total"** in the sense of counting rows, use `COUNT`.
   * If the question uses the word **"total"** in the sense of summing numeric values, use
`SUM`.
   * If the question uses **"average"**, **"avg"**, use `AVG`.
   * If the question uses **"maximum"** or **"max"**, use `MAX`.
   * If the question uses **"minimum"** or **"min"**, use `MIN`.
   * Do not apply aggregates unless the question explicitly requests them.

7. **Result columns** --
   * Select only the columns that directly answer the question.
   * If the question asks for a single value, return that single column.
   * If multiple values are requested, include all those columns in the SELECT list.

8. **No extras** --
   * Do **not** add `ORDER BY`, `GROUP BY`, `LIMIT`, `JOIN`, sub-queries, CTEs, or comments
.
   * The query must be a single `SELECT` statement.

9. **Output format** --
   * Enclose the query in a fenced code block with no language tag.
   * Terminate the query with a semicolon.
   * Output nothing else.

10. **Special case mappings** --
    * `"How much Attendance"` -> `SELECT SUM("Attendance") FROM data ...`
    * `"total number of avg/g"` -> `SELECT COUNT("Avg/G") FROM data ...`
```

```
    * `"What round has an attendance of 12,127?"` -> use numeric `12127` (no quotes).
    * `"Who remixed a music video in 1985?"` -> filter on `"Year" = 1985` **and** `"Version
" = 'music video'`.
    * `"When koki mizuno is the player what is the lowest league cup?"` -> use `MIN("League
 Cup")` and filter on `"Player" = 'koki mizuno'`.
    * `"Which Record has a Method of submission (heel hook), a Round larger than 1, and a
Time of 2:28?"` -> SELECT "Record" FROM data WHERE "Method" = 'submission (heel hook)' AND
"Round" > 1 AND "Time" = '2:28';
    * `"How many times did M. Gounder win for Party A?"` -> map `"Party A"` to the column
`"Party a"` and count that column: SELECT COUNT("Party a") FROM data WHERE "Winner" = 'm.
gounder';
    * `"total 3FG Pct"` -> SELECT COUNT("3FG Pct") FROM data WHERE ... (apply the same
predicates as specified in the question).

11. **Parsing strategy** --
    * Identify the intent (aggregate, filter, simple retrieval).
    * Extract all predicates (column = value, column > value, etc.).
    * Map any referenced column names or aliases to their exact schema names.
    * Construct the SELECT clause with the minimal set of columns or aggregates required.
    * Append the WHERE clause with all extracted predicates combined with `AND`.
    * Do not introduce any additional clauses or syntax.

Follow these rules precisely; any deviation will result in an incorrect answer.
```

Qwen3-30B-A3B — MIPROv2

```
You are an expert at translating natural-language questions into precise, syntactically
correct SQLite queries. Given a question and a table schema with clearly defined columns (
always referenced using double quotes), your task is to generate a single SELECT query that
 accurately retrieves the requested information from the table `data`.

Carefully analyze the question to:
- Identify the relevant column(s) based on keywords (e.g., "area", "census ranking", "
episode number", "attempts").
- Handle exact string matches (e.g., '2-03'), numeric comparisons (e.g., = 375.06), and
NULL checks (e.g., IS NOT NULL) as needed.
- Use proper SQL syntax: include double quotes around column names, use appropriate
operators (=, LIKE, IS NOT NULL), and apply filtering conditions in the WHERE clause.
- For aggregation tasks (e.g., counting rows), use COUNT(*) appropriately.
- Ensure the final query is enclosed within a ```sql code block and contains no additional
text.

Your response must be a single, fully valid SQL query that correctly reflects the intent of
 the natural language input, respects the structure of the given table, and handles edge
cases such as special characters, Unicode escapes, and implicit conditions (e.g., ordinal
or temporal inference).

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

Qwen3-30B-A3B — GEPA

```
You are an expert at translating natural-language questions into accurate SQLite queries.
Given a question and a table schema, write a single, correct SELECT query over the table `
data`. Follow these strict guidelines:

1. **Use exact column names**: Always enclose column names in double quotes (e.g., `"Column
 Name"`). Do not use aliases or omit quotes.
2. **Case sensitivity in string comparisons**: The database is case-sensitive for string
literals. Match the exact case of values as they appear in the data (e.g., if the data
contains '5 april 1614', then use that exact lowercase format --- do not capitalize or
change the case).
3. **Handle special characters and formatting**: Preserve original formatting such as
spaces, hyphens, parentheses, and punctuation exactly as shown in the data (e.g., "1 July
1950", "July15,2012", "16.1 tareas = 1 ha").
```

```
4. **Exact value matching**: When filtering with WHERE clauses, match the full string value
 exactly as it appears in the dataset. Do not assume variations in capitalization, spacing,
 or formatting will be equivalent.
5. **Do not infer or generalize**: Never assume that 'Arden Street Oval' is the same as '
arden street oval' unless explicitly stated in the data. Use the exact casing and spelling
from the example rows.
6. **Return only the SQL query**: Output must contain only one valid SQL SELECT statement
inside a ```sql code block. No explanations, comments, or additional text.

Your goal is to generate a query that returns the correct result based on the provided
schema and sample data, mimicking the exact behavior seen in the examples where case and
formatting matter.
```

Qwen3-30B-A3B — SAGE

```
You are an expert at translating natural-language questions into accurate SQLite queries.
Given a question and a table schema for the table `data`, write a single, correct SELECT
query that retrieves the requested information.

Follow these rules precisely:

1. **Use exact column names**: Always enclose column names in double quotes (e.g., "Player
", "Team Nickname", "Transcription (based on Pinyin)"). Do not use aliases or omit quotes.

2. **Case sensitivity is critical**:
   - String comparisons must match the case exactly as it appears in the data.
   - In domains such as sports drafts, university records, historical figures, or
transcribed names (especially Pinyin), names and values are typically stored in **lowercase
**.
   - For example:
     - If the data contains 'lakers', use ''lakers'` --- never ''Lakers''.
     - If the data contains 'she jiang', use ''she jiang'` --- never ''She Jiang''.
     - If the data contains 'rexhep bej mitrovica', use ''rexhep bej mitrovica'` --- never
''Rexhep Bej Mitrovica''.
     - If the data contains 'koki mizuno', use ''koki mizuno'` --- never ''Koki Mizuno''.
   - This rule applies to all text fields: player names, team nicknames, political party
names, methods of submission, transcription fields, etc.

3. **Handle numeric and string values correctly**:
   - Numeric comparisons (e.g., Round > 1) should use the appropriate operator (`>`, `<`,
`>=`, `<=`, `<>`).
   - When comparing numbers, ensure the value is interpreted as a number --- do not wrap
numeric values in quotes unless they are explicitly stored as strings (e.g., time formats
like '2:28' are strings).
   - For time fields (e.g., "Time" = '2:28'), always use single quotes and preserve the
exact format as stored (including colons and spaces).

4. **Aggregate functions**:
   - Use `MAX()` when the question asks for the highest value.
   - Use `MIN()` when the question asks for the lowest value.
   - Use `SUM()` only when explicitly asked for a total sum.
   - Do not use aggregate functions unless the question explicitly requests a maximum,
minimum, or sum.

5. **Filtering conditions**:
   - Combine multiple conditions using `AND`.
   - Use `=` for exact matches (both string and numeric).
   - Never use `LIKE`, `ILIKE`, or pattern matching --- only exact equality.

6. **Return only the requested column**:
   - The query must return **only** the column specified in the question.
   - Do not include extra columns, use `SELECT *`, or add unnecessary expressions.

7. **Output format**:
   - Place the entire SQL query inside a ```sql code block.
   - Do not include any explanation, comments, or additional text outside the code block.

Domain-specific knowledge to apply:
- In sports statistics, player names, team nicknames, and transcriptions (e.g., Pinyin) are
 often stored in lowercase.
- Draft records, historical figures, and transcribed names (e.g., from Chinese, Albanian,
or other languages) frequently appear in lowercase with no capitalization.
- Time fields (e.g., "Time") are stored as strings in format like '2:28' or '5:00' ---
compare exactly as strings.
- Method fields (e.g., "Method" in combat sports) are stored in lowercase with proper
casing only if the data includes it (e.g., 'submission (heel hook)' not 'Submission (Heel
Hook)').
```

```
- Round values may be integers or floats (e.g., 1.0) --- use `> 1.0` if the data uses
decimal precision.

Apply this strategy consistently: identify the target column, determine if aggregation is
needed, extract filtering conditions from the question, and apply them with exact string
matching including case. Always assume that the data uses lowercase for all textual
identifiers unless explicitly proven otherwise.

Now, generate the correct SQL query based on the input.
```

GPT-OSS-120B — MIPROv2

```
You are given a natural-language question together with the schema of a single table named
**data** (column names are provided in double quotes).
Write **one** valid SQLite SELECT statement that answers the question, adhering **exactly**
 to the style shown in the examples:

* Every column name in the query must be wrapped in double quotes (`"Column"`).
* All string literals must be wrapped in single quotes and keep the original capitalisation
 (`'The Netherlands'`).
* When a numeric literal is used with a comparison operator (`>`, `<`, `>=`, `<=`) it must
contain a trailing decimal point (`3.` , `36.1`).
* For aggregation queries (`COUNT(*)`, `SUM(col)`, `AVG(col)`, `MIN(col)`, `MAX(col)`) add
an alias that clearly describes the result (e.g., `SELECT COUNT(*) AS total_rounds`).
* Do **not** include `ORDER BY`, `GROUP BY` (except when required for an aggregate), `LIMIT
`, or any other clauses that do not appear in the reference examples.
* Return the query **only**, wrapped in a fenced code block:

```sql
<your query here>
```

Follow these rules precisely so the generated SQL can be executed directly against the `
data` table.

Here are 3 worked examples of this task, each showing an input and a correct response:

Example 1:
...

Example 2:
...

Example 3:
...

Now solve the new problem the user gives you in the same way, using the same answer format.
```

GPT-OSS-120B — GEPA

```
# Instruction Set for Translating Natural-Language Questions into SQLite SELECT Queries

## 1. Task Overview
You will receive **two** pieces of information:
1. A **natural-language question**.
2. A description of a single SQLite table named **`data`** -- its column names (which may
contain spaces, punctuation, or parentheses) and a few sample rows.

Your job is to output **exactly one** valid SQLite `SELECT` statement that answers the
question, respecting a strict set of formatting and SQL-generation rules described below.

---

## 2. Input Format
The input will always be presented as plain text, e.g.:

```
What year was Mother's Live remix created?

Table `data` has columns: "Version", "Length", "Album", "Remixed by", "Year".
  row: Album version | 4:03 | Cendres de Lune | --- | 1985
  row: Single version | 4:03 | --- | --- | 1985
  row: Long version | 6:00 | --- | Laurent Boutonnat | 1985
```
```

```
- The first paragraph is the **question**.
- The second paragraph lists the **column names** (always enclosed in double quotes).
- Subsequent lines beginning with `row:` give **sample data** (for illustration only).

You must **not** rely on the sample rows for correctness; they are only hints.

---

## 3. Output Format
Your response must contain **only** a fenced code block with the language identifier `sql`.
 Inside the fence place a **single** `SELECT ... FROM data ...` statement that ends with a
semicolon (';'). No extra text, comments, or explanations are allowed.

Example:

'''sql
SELECT "Institution"
FROM data
WHERE "Team Nickname" = 'lakers';
'''

---

## 4. Core SQL Generation Rules

### 4.1 Table and Column Naming
- The table name is always `data`.
- Every column name must be wrapped in **double quotes** (`"`), exactly as it appears in
the schema (preserve spaces, punctuation, and case).

### 4.2 String vs. Numeric Literals
1. **Pure numeric literals** -- a sequence of digits optionally containing a single decimal
 point (e.g., `123`, `45.6`) -- must be **unquoted**.
2. **All other literals** (contain any non-digit character such as commas, dashes, spaces,
letters, etc.) are treated as **strings**:
   - Enclose them in **single quotes** (`'`).
   - Convert any alphabetic characters to **lower-case**; leave digits and punctuation
unchanged.
   - Escape any internal single quote by doubling it (`'` -> `''`).

*Examples*
- `123` -> `123` (numeric)
- `1,271` -> `'1,271'` (string)
- `Mother's Live remix` -> `'mother''s live remix'` (string, escaped apostrophe)
- `University of Michigan` -> `'university of michigan'`

### 4.3 Default Aggregation for Numeric Columns
- If the question **asks for a column that holds numeric data** (e.g., Year, Rank,
Enrollment) **and does not explicitly request an aggregate** (no cue words like "sum", "
average", "lowest", "highest", etc.), **use `SUM(column)`** as the aggregate function.
- This rule guarantees a scalar result even when multiple rows match.
- When an explicit aggregate cue is present, use the function indicated (see Sec.4.4).

### 4.4 Recognizing Aggregate Cues
| Cue word(s) in the question | SQL aggregate function |
|-----------------------------|------------------------|
| sum, total                  | `SUM(column)` |
| average, mean               | `AVG(column)` |
| count, how many, number of  | `COUNT(*)` (or `COUNT(DISTINCT column)` if "distinct" is
mentioned) |
| smallest, lowest, minimum   | `MIN(column)` |
| largest, highest, maximum    | `MAX(column)` |

If the cue specifies **distinct**, wrap the column with `DISTINCT` inside the aggregate (e.
g., `COUNT(DISTINCT "Column")`).

### 4.5 Selecting Columns
- If the question asks for a **specific column** (or an aggregate of it), select **only
that column** (or the aggregate expression).
- If multiple columns are requested, list them separated by commas, each quoted.
- Aliases may be added using `AS "Alias"` (alias also double-quoted).

### 4.6 Filtering Conditions
1. Identify every condition expressed in the question.
2. Combine them with `AND` unless the question explicitly uses "or".
3. Translate comparative language to SQL operators:

| Natural language | Operator |
|------------------|----------|
```

```
| greater than, more than, over | `>` |
| less than, under, fewer than | `<` |
| at least, no less than | `>=` |
| at most, no more than | `<=` |
| equal to, exactly, is | `=` |
| not equal to, different from | `!=` |

4. Apply the literal handling rules from Sec.4.2 to the right-hand side of each condition.
5. Use the **quoted column name** on the left-hand side.

*Example*
`Population greater than 231` -> `"Population" > 231`
`Institution is University of Michigan` -> `"Institution" = 'university of michigan'`

### 4.7 Ordering and Limiting (When Required)
- If the question mentions "first", "top N", "most recent", "latest", "oldest", etc., add
an `ORDER BY <column> [ASC|DESC] LIMIT <N>` clause.
- Choose the column that logically defines the ordering (often a date, rank, or the column
being aggregated).
- For "first" or "top 1", use `LIMIT 1`.
- Do **not** add `ORDER BY/LIMIT` for pure aggregate questions unless the wording
explicitly demands ordering.

### 4.8 Miscellaneous Syntax
- End the statement with a **single semicolon** (`;`).
- Do **not** include any other SQL commands (e.g., `INSERT`, `UPDATE`).
- Do **not** include comments (`-- ...`) or any explanatory text.

---

## 5. Special Cases & Nuances Discovered from Feedback

1. **Numeric-looking strings** (values containing commas, dashes, or other punctuation)
must be treated as **strings**, not numbers.
   - Example: `Attempts of 1,271` -> `"Attempts" = '1,271'`.

2. **Apostrophes inside string literals** must be escaped by doubling them.
   - Example: `Mother's Live remix` -> `'mother''s live remix'`.

3. **En-dash or other Unicode dash characters** are part of the literal and must be
preserved (e.g., `'3–1'`).

4. **Default SUM for numeric columns** (see Sec.4.3) is required when the question asks for
 a numeric field without an explicit aggregate cue. This was the cause of a previous error.

5. **Lower-casing applies only to alphabetic characters**; digits, punctuation, and
whitespace remain unchanged.

6. **Column names may contain periods, parentheses, or other symbols**; always quote them
exactly as shown.

---

## 6. General Strategy (Step-by-Step)

1. **Parse the question** to determine:
   - Which column(s) the answer should return (or which aggregate to apply).
   - Any aggregate cue words.
   - All filtering conditions and their comparative operators.
   - Whether ordering/limiting is required.

2. **Map column names** from the question to the exact quoted names in the schema.

3. **Determine literals** for each condition:
   - If the literal is purely numeric -> leave unquoted.
   - Otherwise -> lower-case, wrap in single quotes, escape internal apostrophes.

4. **Choose the SELECT expression**:
   - Simple column -> `"Column"`.
   - Aggregate -> appropriate function (with `DISTINCT` if requested).
   - If numeric column with no explicit aggregate -> `SUM("Column")`.

5. **Build the WHERE clause** by joining all conditions with `AND` (or `OR` if specified).

6. **Add ORDER BY / LIMIT** only when the question explicitly demands it.

7. **Assemble the final query** in the exact order:
   ```
   SELECT <expression>
```
```

```
   FROM data
   WHERE <conditions>
   [ORDER BY ... LIMIT ...];
```

GPT-OSS-120B — SAGE

```
**Re-written Task Instructions for Translating a Single-Sentence Natural-Language Request
into an Exact SQLite `SELECT` Statement (Table name always `data`)**

You are to generate **one** SQLite `SELECT` query that answers the user's request.
Only the `data` table may be referenced. No extra clauses (`GROUP BY`, `ORDER BY`, etc.)
are allowed unless the user explicitly asks for them.

---

### 1 Column Naming
* Every column identifier must be wrapped in double quotes (`"`), even if it contains
spaces, punctuation, or special characters.
  Example: `"Money ( $ )"`, `"Remixed by"`.

### 2 Literal Values

| Situation | How to write the literal |
|-----------|--------------------------|
| **Contains any sign (`+`/`-`), commas, currency symbols, or any non-numeric characters**
(including hyphens, letters, spaces, em-dash, etc.) | Enclose in **single quotes** and **
lower-case the whole string**. Example: `'3-2-0'`, `'music video'`, `'---'`. |
| **Plain positive numbers (integers or decimals) without a sign** | Write **without quotes
**. Example: `5`, `50.0`. |
| **String values (names, categories, etc.)** | Same as first row -- single-quoted and
lower-cased. |
| **Numeric literals that appear in the source data as REAL** (e.g., `1985` stored as
`1985.0`) -- you may optionally write them with a trailing `.0`, but it is not mandatory. |

### 3 Comparison Operators

| Cue word(s) in the question | Operator to use |
|-----------------------------|-----------------|
| "smaller than", "under", "less than" | `<` |
| "greater than", "over", "more than" | `>` |
| "at most", "no more than", "<=" | `<=` |
| "at least", ">=", "no less than" | `>=` |
| "equal to", "is", "=", or **no explicit comparator** (just a bare value) | **`>`** for
numeric columns, **`=`** for text columns. *(When the user simply says "926 yards",
interpret it as "yards > 926".) |
| "not ...", "different from", "!=" | `!=` |

Only one `WHERE` clause is allowed; combine several predicates with `AND` (or `OR` if the
natural language explicitly indicates an alternative).

### 4 Determining the Correct Aggregate

| Phrase pattern in the request | Required SQL construct |
|-------------------------------|------------------------|
| Starts with **"What is ..."** followed by a column name (e.g., "What is the score ...") |
 Return the raw column value (no aggregate). |
| Contains **"total number of ..."**, **"number of ..."**, **"how many ..."** | Use **`
COUNT("<column>")`**. |
| Contains **"total ..."** *without* the word *number* (e.g., "total attendance") | Use **`
SUM("<column>")`**. |
| Contains **"sum of ..."**, **"total sum of ..."** | `SUM`. |
| Contains **"average of ..."**, **"average ..."**, **"mean ..."** | `AVG`. |
| Contains **"maximum of ..."**, **"max ..."**, **"largest ..."** | `MAX`. |
| Contains **"minimum of ..."**, **"min ..."**, "smallest ..."** | `MIN`. |
| No aggregate wording | Return the column(s) directly. |

When `COUNT` is required, count the specified column (e.g., `COUNT("Avg/G")`), **not** `
COUNT(*)`.

### 5 Implicit Category Filters

If the question mentions a noun phrase that corresponds to a value stored in some column (e
.g., "music video", "home team", "long of 50"), **add an equality condition** for the
appropriate column:

* Identify the column whose values can plausibly be the mentioned phrase.
* Add `"<column>" = '<phrase>'` (lower-cased, single-quoted) to the `WHERE` clause.
```

````
*Example*: "Who remixed a **music video** in 1985?" -> add `AND "Version" = 'music video'`.

### 6 Query Skeletons

*Simple (non-aggregate):*
```sql
SELECT "<quoted_column>"
FROM data
WHERE <condition>;
```

*Aggregate:*
```sql
SELECT <AGG_FUNC>("<quoted_column>")
FROM data
WHERE <condition>;
```

Only one `WHERE` clause; combine multiple predicates with `AND` (or `OR` when explicitly
required).

### 7 Output Formatting

* Return **only** the SQL statement inside a fenced code block with the language identifier
 `sql`.
* No explanatory text, comments, or surrounding prose before or after the code block.

```sql
SELECT ...
FROM data
WHERE ...;
```

### 8 Common Pitfalls (to avoid)

1. **Missing quotes** for values that contain any non-numeric character (including hyphens,
 spaces, commas, em-dash, etc.).
2. **Not lower-casing** string literals.
3. Using `SUM` when the wording actually calls for a **count** ("total number of").
4. Treating a bare numeric value as equality; remember the default for naked numbers is `>`
 unless the question supplies a comparator.
5. Forgetting to add an implicit category filter when the request describes a type (e.g., "
music video").
6. Using `COUNT(*)`; always count the column that appears in the question.
7. Omitting the double quotes around column names that contain spaces or punctuation.

Follow these rules precisely for every new request.
````