

Beyond Verified Answers: Solver-Informed Self-Distillation for Bootstrapping Operations Research Language Models

Rui Zhu^{1,*} Minglong Cao^{2,*} Chenyu Zhou^{1,*} Jianghao Lin^{1,†} Dongdong Ge¹

¹Shanghai Jiao Tong University, Shanghai, China

²Fudan University, Shanghai, China

{jameszhu, chenyzhou, linjianghao, ddge}@sjtu.edu.cn mlcao25@fudan.edu.cn

*Equal contribution †Corresponding author

Abstract Modern large language models (LLMs) can translate natural-language descriptions into operations research (OR) formulations. Post-training techniques including reinforcement learning and on-policy self-distillation have further improved this capability. However, three limitations remain in training LLMs for OR formulations. First, training commonly relies on synthetic formulations validated by human experts or stronger models, constraining scalable supervision. Second, credit assignment is either coarse or costly: outcome rewards score an entire trajectory without locating the responsible modeling decision, whereas process-level supervision requires an additional evaluator. Third, privileged self-distillation can induce style mismatch by using solver context unavailable at deployment. We find that a model can improve from solver-artifact feedback generated by its own rollouts, making self-distillation a practical, evaluator-free source of dense supervision. Therefore, we propose **SOLID**—**S**olver-Informed **O**n-Policy **L**earn**I**ng through **S**elf-**D**istillation, a novel framework for self-improving OR language models without verified answers or external evaluators. SOLID executes candidate programs from multiple rollouts, clusters their objectives, and selects a majority-group artifact as a pseudo-reference. The model then performs updates using group-relative advantages and dense self-supervision signals. Across multiple OR benchmarks, SOLID improves solution accuracy for both general-purpose and OR-tuned models over outcome-only group-relative training. These results show that solver artifacts can support scalable self-improvement without trusted answers.

🔗 **Code:** <https://github.com/AIOR-Research/SOLID>

🤖 **Model Weights:** <https://huggingface.co/AIOR-Research/SOLID-Qwen3>

1. Introduction

Large language models (LLMs) can derive mathematical formulations and executable solver programs from natural-language operations research (OR) problem descriptions. Inference-time search and reinforcement learning (RL) have improved this capability [8, 15, 18, 23, 30]. Reliable OR modeling remains challenging, however, because the variables, objective, constraints, and code must jointly represent the intended decision problem [11]. Despite this progress, these RL related paradigms still face three key bottlenecks in the OR setting.

The first obstacle is **the cost of scalable supervision**. Existing pipelines expand curated seed formulations into synthetic training examples and validate them with trusted solutions, stronger models, or learned evaluators [8, 15, 30]. In addition, validation still requires OR expertise because equivalent formulations may differ substantially, while an incorrect formulation may execute and return a plausible objective. This dependence limits learning from problems for which descriptions and solver access are available but verified answers are not.

The second obstacle is **accurate credit assignment within a single trajectory**. This is challenging in OR, since reasoning typically spans variable definition, objective formulation, constraint construction, and code generation.

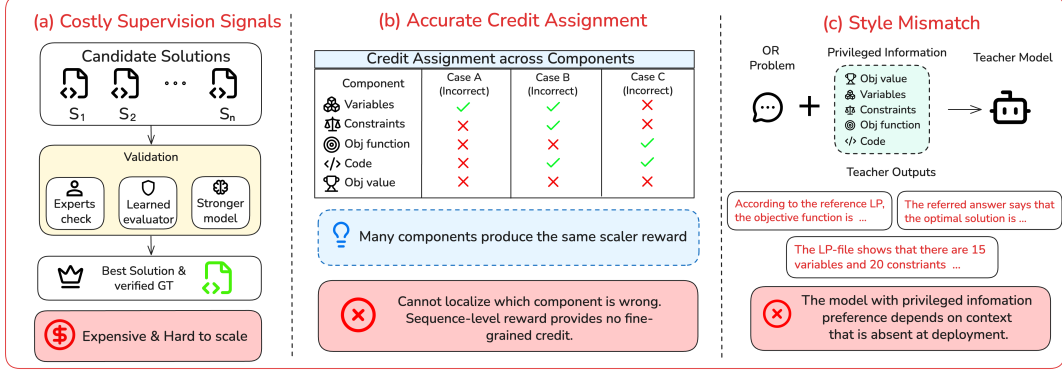


Figure 1: Three challenges motivating SOLID: costly supervision, coarse sequence-level credit assignment, and privileged-context style mismatch.

OR research has refined sequence-level signal by evaluating formulation components separately, deriving verifiable feedback from generated code and solver artifacts, and learning process reward models for intermediate reasoning [4, 18, 30]. These approaches provide finer credit, but require component annotations, predefined verification signals, or another learned evaluator. On-policy self-distillation (OPSD) provides an economical alternative for assigning dense credit: the same model re-evaluates its trajectory under privileged information, without a stronger teacher [10, 27].

The third obstacle is **style mismatch**, a form of information leakage in which the contextual policy conditions its predictions on evidence that is unavailable at deployment time [13, 16]. This issue is particularly pronounced in operations research. When a correct solver artifact is provided as privileged information, the privileged policy may rename variables, omit intermediate derivations, or simply restate the artifact itself. Consequently, directly matching the privileged policy’s distribution conflates superficial changes in form and degrading performance.

To address all obstacles, we propose SOLID (Solver-Informed On-Policy LearnIng through Self-Distillation), which converts solver information from the same rollout group into both label-free sequence supervision and token-level supervision. Solver-outcome supplies the group-relative learning signal, while one majority-group solver artifact provides privileged context for self-distillation. A sample mask restricts self-distillation only to structurally mismatched variable, objective, constraint, and code sections. This closed loop requires neither verified answers nor an external evaluator and mitigates style mismatch.

Our contributions are threefold:

- We formulate OR language-model post-training *without* verified answers as a closed self-bootstrapping loop over executable on-policy rollouts and solver-outcome consensus, requiring no external evaluator.
- We introduce complementary sequence-level and token-level supervision from the same rollout group. Solver-outcome agreement drives group-relative learning, while self-distillation with an LP-structured mask blocks potentially harmful updates for plausible or unrelated reasoning steps.
- SOLID consistently outperforms self-distillation and label-free reinforcement learning methods

across challenging benchmarks. Its effectiveness on base and post-trained models further demonstrates its broad applicability and potential for scalable OR post-training without verified answers.

2. Related Work

Language Models for Optimization Modeling. Early auto-formulation systems cast natural-language descriptions into structured linear-programming (LP) components and emphasized constrained generation and human validation [17, 18]. Subsequent work broadened the target to end-to-end formulation, code generation, and solving. Agentic approaches organize these activities through external tools and interaction protocols, within a broader landscape of memory, skills, and harness engineering [28]. OptiMUS uses modular decomposition and iterative solver-based correction, while BPP-Search explores alternative formulations through tree search [1, 23]. Agora-Opt combines solver-outcome-based decentralized debate with persistent memory to support training-free improvement in optimization modeling [14]. OR-Space extends evaluation to model construction, revision, and grounded explanation in persistent industrial workspaces [29].

Solver-Informed Supervision. Solver-informed supervision uses executable programs and optimization artifacts to filter training data or construct learning signals. Existing OR post-training pipelines typically build instruction data from synthetic formulations and develop trusted answers from solver information [8, 9, 15]. More recent methods move solver information into the learning signal: SIREL derives verifiable rewards from generated code and LP artifacts; StepORLM adds a co-evolved generative process reward model; and MURKA combines semantic and execution rewards with an external teacher and a checker agent [4, 24, 30]. Although these approaches make supervision more task-grounded, their supervision signals still rely on trusted sources or external advanced models. Thus, how to derive dense supervision from solver feedback without verified answers remains an open question.

Self-Supervised Post-Training. Self-supervised post-training can derive learning signals from a model’s own on-policy rollouts. At sequence level, Test-time Reinforcement Learning (TTRL) turns majority agreement into a group-relative reward for training without ground truth [21, 32]. At the token level, on-policy self-distillation (OPSD) re-scores a rollout using the same policy conditioned on privileged information, providing dense guidance without a stronger teacher [7, 10, 27]. However, this contextual signal is not always reliable: privileged information may alter the notation, ordering, or reasoning style, causing self-distillation to introduce style mismatch and degraded performance [12, 16]. SOLID connects these two branches by retaining TTRL’s group-relative outcome learning while applying self-distillation only to the segments of the reasoning traces where errors occur, thereby mitigating the **style mismatch** problem.

3. Preliminaries

Let x be an unlabeled OR problem and $y = (y_1, \dots, y_T)$ a structured response ending in executable solver code. Following recent LLM-based OR reasoning pipelines [4, 30], we view y as a progression from problem interpretation to decision variables, objective, constraints, mathematical model, and solver code. The rollout policy samples $\{y_i\}_{i=1}^N \sim \pi_{\text{old}}(\cdot \mid x)$. An external Python executor then runs each generated program and returns an invalid status or an optimal objective $\hat{z}_i$ together with its LP artifact. A simplified LP artifact has the following solver-readable form:

```

Minimize
    50 x + 75 y + 20 z
Subject To
    capacity: 50 x + 75 y + 20 z <= 20000
    balance: -2.33 x - y + z >= 0
Bounds
    x >= 50
Generals
    x y z
End
    
```

Step-level credit assignment. Step-level methods refine trajectory-level supervision by assigning feedback to intermediate decisions. StepOPSD is one example. It decomposes agent trajectories into action-centered segments and redistributes hindsight-conditioned distillation signals at this granularity [26]. OR modeling provides a natural step structure because a formulation is assembled through decisions about variables, objectives, constraints, mathematical models, and executable code [8, 18]. StepORLM makes this structure explicit by applying generative process supervision to intermediate OR modeling steps [30]. This structure motivates localizing credit at the step level.

Reinforcement learning without labels. Following common reinforcement-learning practices in mathematical reasoning and OR, we adopt Test-time Reinforcement Learning (TTRL), an algorithm for reinforcement learning without labels, as our baseline for unlabeled OR problems. Its implementation is based on Group Relative Policy Optimization (GRPO). GRPO normalizes group rewards and applies a PPO-style clipped update without a learned value model [20, 21]. Following TTRL [32], solver-outcome agreement supplies the vote reward, while format and execution rewards enforce structured output and solver validity [4]:

$$r_i = r_i^{\text{fmt}} + r_i^{\text{exec}} + r_i^{\text{vote}}, \quad A_i = \frac{r_i - \mu_r}{\sigma_r + 10^{-8}},$$

and the corresponding clipped loss without Kullback–Leibler (KL) divergence is

$$\begin{aligned} \mathcal{L}_{\text{GRPO}} &= -\mathbb{E}_{i,t} [\min\{\rho_{i,t} A_i, \bar{\rho}_{i,t} A_i\}], \\ \bar{\rho}_{i,t} &= \text{clip}(\rho_{i,t}, 1 - \epsilon, 1 + \epsilon), \\ \rho_{i,t} &= \frac{\pi_{\theta}(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{old}}(y_{i,t} \mid x, y_{i,<t})}. \end{aligned}$$

Every token shares A_i , so this **sequence-level supervision** cannot locate the responsible modeling choice.

On-policy self-distillation. For the same rollout y_i , let $\tilde{x}_i$ denote a privileged-context version of x . OPSD uses the rollout policy as a fixed contextual teacher while the updated policy remains the deployable student [27]:

$$\begin{aligned} \ell_{i,t}^T &= \log \pi_{\text{old}}(y_{i,t} \mid \tilde{x}_i, y_{i,<t}), \\ \ell_{i,t}^{\theta} &= \log \pi_{\theta}(y_{i,t} \mid x, y_{i,<t}), \quad \Delta_{i,t} = \ell_{i,t}^T - \ell_{i,t}^{\theta}. \end{aligned}$$

Using the same sampled reverse-KL estimator as SOLID [19], whole-response OPSD minimizes

$$\mathcal{L}_{\text{OPSD}} = \mathbb{E}_{i,t} [\phi_{k_3}(\Delta_{i,t})], \quad \phi_{k_3}(\Delta) = e^{\Delta} - \Delta - 1.$$

Thus, OPSD provides dense token-level supervision but applies it uniformly to the response and does not include the group-relative objective. In OR, the majority-LP pseudo-reference can supply $\tilde{x}_i$ and expose the model’s correction preferences without another evaluator. Because uniform re-scoring suffers from **style mismatch**, SOLID uses the binary LP-structured mask to localize supervision signals [16].

4. Methodology

Overview. SOLID follows the three components summarized in Figure 2. **On-policy inference** samples N responses under a fixed structured prompt. A **solver executor and information extractor** execute the generated code and turn their objective values and LP artifacts into a majority-LP pseudo-reference and candidate-reference LP-structure differences. **Solver-informed learning** uses solver-outcome agreement for sequence-level advantage and the reference artifact for self-distillation.

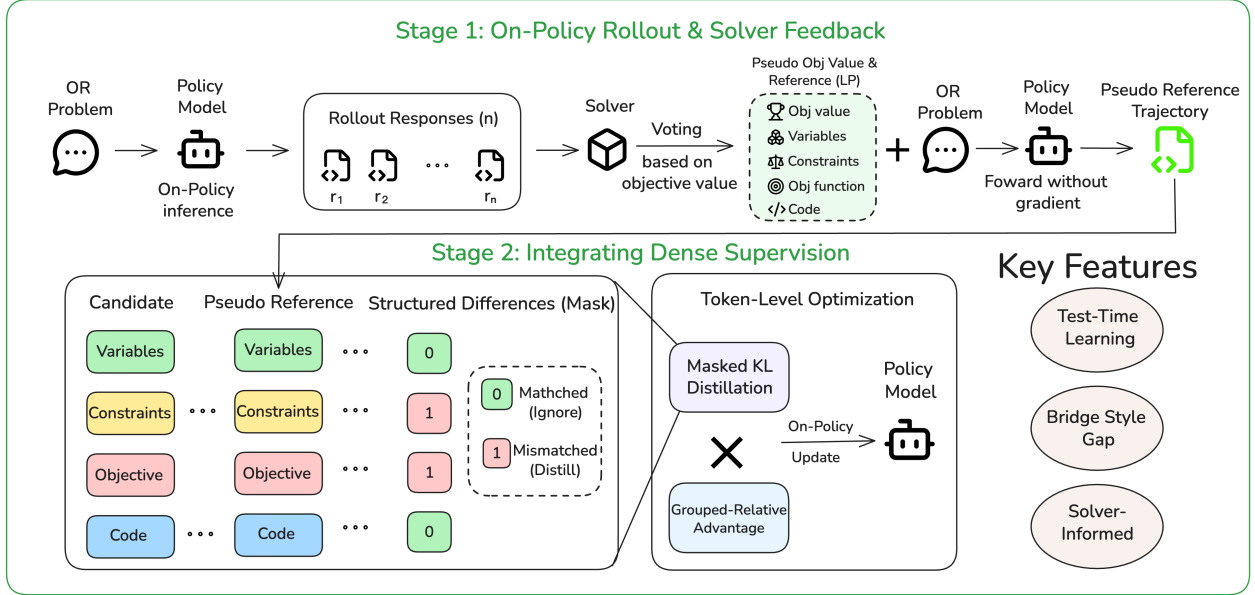


Figure 2: Overview of SOLID. The policy first generates on-policy trajectories whose programs are evaluated and grouped by solver feedback, yielding group-relative rewards and a majority-group solver artifact. The same policy then uses this artifact as privileged context, while an LP-structured mask restricts self-distillation to structurally mismatched variable, constraint, objective, and code sections. The masked token-level signal and sequence-level reward jointly update the deployable policy.

4.1. SOLID

On-policy inference. A fixed response template makes each rollout structurally separable: it asks the model to reason step by step in ordered sections with declared modeling roles. The system message supplies the format instruction, while the user message contains the OR problem. The reference field is empty during rollout and contains the majority-LP pseudo-reference only during contextual re-scoring. This design is not tied to a particular prompt template: for any separable response schema, solver-grounded sections can be associated with the corresponding artifact evidence and masked according to candidate-reference differences. In our implementation, we instantiate this principle with a nine-stage response template; the complete prompt is provided in supplementary material. An abridged template is:

Table 1: Mapping LP signatures to the StepORLM response template.

Compared LP signature	Response section
Variables, types, bounds, or columns	3. Decision Variables
Objective sense or coefficients	4. Objective Function
Row coefficients, senses, or RHS	5. Constraints
Complete normalized LP emitted by the candidate code	9. Python Code Using gurobipy

```

SYSTEM:
{FORMAT INSTRUCTION}
Reason step by step in separated sections (<step>...</step>).
Use the solver-information reference only in allowed sections.
{REFERENCE LP}
(contextual scoring only; empty during rollout)
USER:
{OR PROBLEM}
    
```

The prompt mapping identifies where solver evidence is applicable; the binary LP-structured mask itself is determined solely by candidate–reference artifact comparison.

Solver executor and information extractor. For each rollout, the executor records execution status, objective $\hat{z}_i$, and LP artifact a_i . After normalizing objective direction, executable objectives within 10^{-6} form a vote cluster. Let C^* be the largest cluster and $\hat{z}_{\text{vote}}$ its median. The executable rollout closest to $\hat{z}_{\text{vote}}$ supplies the majority-LP pseudo-reference s^* . If none executes, $\hat{z}_{\text{vote}} = \perp$ and no reference is constructed. The vote reward is

$$r_i^{\text{vote}} = \mathbf{1}[i \in C^*].$$

The extractor next compares each candidate artifact with s^* . It canonicalizes variable identities and LP record order, then builds signatures for variables (types, bounds, and constraint-matrix columns), the objective (sense and coefficients), constraints (sorted rows of coefficients, senses, and right-hand sides), and the complete normalized LP emitted by the code. Table 1 illustrates how these signatures map to the StepORLM response template [30]; other implementations may adapt the mapping to their response schema and available solver information. For the mapped sections $\mathcal{K} = \{3, 4, 5, 9\}$,

$$d_{i,k} = \mathbf{1}[F_k(a_i) \neq F_k(s^*)], \quad k \in \mathcal{K},$$

where F_k extracts the corresponding signature. If a_i is missing, invalid, or unseparable, only $d_{i,9}$ is set to one because only the code failure is observable. The extracted evidence is therefore

$$E_i = (\hat{z}_{\text{vote}}, s^*, d_i).$$

Let $\kappa_i(t)$ denote the response section containing token t . The sample gate and final LP-structured mask are

$$u_i = \mathbf{1}[\hat{z}_{\text{vote}} \neq \perp] \mathbf{1}[i \notin C^*],$$

$$b_{i,t} = \begin{cases} u_i d_{i,\kappa_i(t)}, & \kappa_i(t) \in \mathcal{K}, \\ 0, & \text{otherwise.} \end{cases}$$

Thus, majority-cluster responses receive GRPO only, while non-majority responses can receive self-distillation only in LP-mismatched mapped sections.

Algorithm 1 SOLID on unlabeled OR prompts

Require: Prompts $\mathcal{D}$, policy π_θ , rollouts N , coefficient β

- 1: **for** each minibatch $\mathcal{B} \subset \mathcal{D}$ **do**
- 2: Set rollout snapshot $\pi_{\text{old}} \leftarrow \pi_\theta$.
- 3: **for** each $x \in \mathcal{B}$ **do**
- 4: **Inference:** sample $\{y_i\}_{i=1}^N \sim \pi_{\text{old}}(\cdot \mid x)$.
- 5: **Executor:** extract and run code; record status, $\hat{z}_i$, and a_i .
- 6: **Extractor:** form $C^\star$, $\hat{z}_{\text{vote}}$, $s^\star$, rewards, and A_i .
- 7: **for** each rollout i **do**
- 8: **if** $\hat{z}_{\text{vote}} \neq \perp$ and $i \notin C^\star$ **then**
- 9: Compare $F_k(a_i)$ with $F_k(s^\star)$; form $b_{i,t}$.
- 10: Cache $\ell_{i,t}^T$ by scoring y_i under $\tilde{x}_i$.
- 11: **else**
- 12: Set $b_{i,t} = 0$ for all t .
- 13: **end if**
- 14: **end for**
- 15: **end for**
- 16: Compute $\mathcal{L}_{\text{GRPO}}$ and $\mathcal{L}_{\text{KL}}$.
- 17: Update θ using $\mathcal{L}_{\text{GRPO}} + \beta \mathcal{L}_{\text{KL}}$.
- 18: **end for**

Solver-informed learning objective. The majority objective and majority-LP pseudo-reference are inserted into a context-augmented prompt,

$$\tilde{x}_i = g_{\text{OR}}(x, \hat{z}_{\text{vote}}, s^\star).$$

The rollout policy re-scores the same response without gradients under $\tilde{x}_i$, while the updated student sees only the original problem:

$$\begin{aligned} \ell_{i,t}^T &= \log \pi_{\text{old}}(y_{i,t} \mid \tilde{x}_i, y_{i,<t}), \\ \ell_{i,t}^\theta &= \log \pi_\theta(y_{i,t} \mid x, y_{i,<t}). \end{aligned}$$

This same-policy comparison provides token-level credit without verified answers or another evaluator. Because privileged LP context can also change notation, ordering, or derivation detail, uniform self-distillation suffers from **style mismatch**. The sample gate avoids KL across consensus-equivalent responses, and the LP-structure gate prevents contextual changes in matched or unrelated sections.

Let $\Delta_{i,t} = \ell_{i,t}^T - \ell_{i,t}^\theta$. We use the non-negative k_3 estimator [19],

$$\phi_{k_3}(\Delta) = e^\Delta - \Delta - 1, \quad \mathcal{L}_{\text{KL}} = \mathbb{E}_{i,t} [b_{i,t} \phi_{k_3}(\Delta_{i,t})].$$

The rollout importance ratio appears only in the GRPO/PPO policy loss and does not multiply $\mathcal{L}_{\text{KL}}$. If no majority-LP pseudo-reference exists, then $b_{i,t} = 0$ and the KL term vanishes. The complete learning objective is

$\mathcal{L}_{\text{SOLID}} = \mathcal{L}_{\text{GRPO}} + \beta \mathcal{L}_{\text{KL}}$

GRPO supplies sequence-level competition, while the masked k_3 term provides LP-structured token-level credit. The following section analyzes how the group advantage anchors these contextual corrections.

5. Anchored Correction under Style Mismatch

Contextual likelihood shifts conflate task correction with **style mismatch** in wording, order, or derivation detail. SOLID separates their roles: the LP-structured mask determines where context may act, while the group advantage determines how strong that preference must be to reverse an update. Proofs are provided in the supplementary material. Let $p_{i,t} = \pi_\theta(y_{i,t} \mid x, y_{i,<t})$, $q_{i,t} = \pi_{\text{old}}(y_{i,t} \mid \tilde{x}_i, y_{i,<t})$, and $\Delta_{i,t} = \log q_{i,t} - \log p_{i,t}$. Here we consider the unclipped scenarios.

Lemma 1 (masked contextual direction). For fixed $q_{i,t}$ and importance ratios,

$$-\nabla_\theta \mathcal{L}_{\text{KL}} = \mathbb{E}_{i,t} \left[\tilde{b}_{i,t} \left(\frac{q_{i,t}}{p_{i,t}} - 1 \right) \nabla_\theta \log p_{i,t} \right].$$

$$\begin{aligned} A_{i,t}^{\text{eff}} &= A_i + \beta b_{i,t} (e^{\Delta_{i,t}} - 1), \\ -\nabla_\theta \mathcal{L}_{\text{SOLID}} &= \mathbb{E}_{i,t} [A_{i,t}^{\text{eff}} \nabla_\theta \log p_{i,t}]. \end{aligned}$$

For small $\Delta_{i,t}$, this becomes

$$A_{i,t}^{\text{eff}} = A_i + \beta b_{i,t} (\ell_{i,t}^T - \ell_{i,t}^S) + O(\beta \Delta_{i,t}^2),$$

which formalizes the log-probability-difference approximation.

Negative sample reinforcement suppresses a sampled response and redistributes mass toward alternatives [31]. Non-majority rollouts are similarly *vote-negative*, though not necessarily incorrect. When $A_i < 0$, the negative samples can therefore suppress a locally useful token.

Proposition 1 (anchored sign reversal). For $A_i < 0$, $b_{i,t} = 0$ gives $A_{i,t}^{\text{eff}} = A_i$. If $b_{i,t} = 1$, let

$$\tau_i = \log \left(1 - \frac{A_i}{\beta} \right) > 0.$$

Then $A_{i,t}^{\text{eff}} \leq A_i$ for $\Delta_{i,t} \leq 0$; $A_{i,t}^{\text{eff}} \leq 0$ for $0 < \Delta_{i,t} \leq \tau_i$; and $A_{i,t}^{\text{eff}} > 0$ only for $\Delta_{i,t} > \tau_i$.

Thus masking removes the KL channel for majority, matched, and unmapped tokens, regardless of their surface-form shift. Within an activated section, A_i is an anchor: weak contextual preferences cannot reverse a negative update, while a strong LP-informed preference can preserve a locally useful token. The LP-structured mask ensures that this occurs only at error-related reasoning stages, thereby preventing reversals caused by **style mismatch**.

6. Experiments

6.1. Experimental Setup

Benchmarks and Metrics. We evaluate on OptMATH [15], MAMO-Complex [9], and IndustryOR (InOR) [8], three particularly demanding benchmarks spanning general mathematical optimization, complex executable modeling, and real-world industrial OR scenarios. These benchmarks require end-to-end semantic, mathematical, and executable correctness rather than surface-form matching or short-answer accuracy. We report majority accuracy $\text{maj}@N$ and $\text{pass}@k$. $\text{Maj}@N$ denotes the accuracy of the answer selected by majority voting among N sampled solutions. Following Chen et al. [3], $\text{pass}@k$ measures the probability that at least one of k sampled solutions is correct and is estimated as

$$\text{pass}@k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}},$$

where n is the total number of generated samples and c is the number of correct samples. We mark a sample as correct if its derived objective value is within 10^{-6} of the ground-truth value. All values reported in the tables are percentages.

Baselines.

- **Base.** We evaluate both Qwen3-4B-Instruct [25] and StepORLM [30], an OR-specialized model built on Qwen3-8B. The corresponding model before task-specific training is reported as **Base**.
- **TTRL.** The outcome-only, group-relative baseline that derives rewards from majority voting over objective values[32].
- **OPSD.** A whole-response self-distillation baseline in which the policy re-scores its own rollouts under privileged LP context, without LP-structured error localization [27]. Unless otherwise stated, OPSD uses the majority-LP solution as the privileged information.
- **SOLID.** Our method combines the TTRL-style group-relative objective with solver-informed, LP-structured masked self-distillation.

Implementation details. For the main training run, we construct a 50,000-instance subset from the publicly released **OptMATH-Train** corpus [2, 15]. Specifically, we generate five independent responses to each candidate problem using Qwen3-8B and retain problems that are solved correctly at least once but no more than three times, i.e., with a success count in $\{1, 2, 3\}$ out of five trials.

Unless otherwise specified, all generated programs in our pipelines are executed using Gurobi [6]. Gurobi provides the solver statuses and artifacts used to determine the LP-structured mask. For StepORLM [30], we retain its native COPT backend [5] to avoid introducing conversion errors between solver APIs.

During training, we sample 24 trajectories per prompt at a temperature of 1.0 with a batch size of 32. We use a GRPO clipping range of 0.2, set the KL coefficient to 10^{-3} , and allow responses of up to 32,768 tokens. Training is conducted on four NVIDIA H100 GPUs (4×H100), and our implementation is built on vERL [22]. Evaluation keeps the same configuration as well and checkpoints are evaluated at the same training steps.

6.2. Main Results

Qwen3-4B-Instruct results. Table 2a compares Base, TTRL, and SOLID under the same training budget. The strongest gains occur on OptMATH, where majority accuracy rises by 9.64 points and $\text{pass}@k$ improves by an average of 6.31 points across $k \in \{1, 2, 4\}$. Improvements also extend across all three $\text{pass}@k$ metrics on InOR. When OptMATH and InOR are aggregated over their 266 problems, SOLID improves majority accuracy by 6.02 points and $\text{pass}@k$ by an average of 4.09 points over TTRL. The results reveal steady improvements of SOLID across datasets.

StepORLM results. Table 2b compares Base, TTRL, and SOLID on an OR-tuned model. Under the same training budget, SOLID improves or matches majority accuracy on all three datasets. It also outperforms TTRL on eight of twelve metrics overall. The largest gains occur on MAMO-Complex, where $\text{pass}@k$ improves by an average of 3.75 points across $k \in \{1, 2, 4\}$. On OptMATH and InOR, the $\text{pass}@k$ results remain broadly comparable to TTRL, with gains at smaller k and slight decreases at larger k .

Table 2: Base and matched-budget results on three datasets. TTRL and SOLID use the same training budget and evaluation protocol. Bold marks the best value within each dataset–metric pair.

(a) Qwen3-4B-Instruct.						(b) StepORLM.					
Dataset	Method	maj@64	pass@1	pass@2	pass@4	Dataset	Method	maj@64	pass@1	pass@2	pass@4
OptMATH	Base	24.70	14.16	21.89	31.33	OptMATH	Base	17.47	10.99	16.67	22.29
OptMATH	TTRL	30.12	19.01	25.29	31.32	OptMATH	TTRL	30.72	17.48	23.64	30.38
OptMATH	SOLID	39.76	24.25	31.62	38.70	OptMATH	SOLID	31.33	18.25	24.40	30.53
MAMO-Complex	Base	35.47	25.00	34.81	44.33	MAMO-Complex	Base	62.56	52.22	65.02	72.91
MAMO-Complex	TTRL	33.50	27.56	34.95	41.52	MAMO-Complex	TTRL	68.97	61.22	68.08	72.26
MAMO-Complex	SOLID	33.00	29.51	35.71	41.34	MAMO-Complex	SOLID	70.44	66.43	71.58	74.79
InOR	Base	52.00	41.56	48.84	55.11	InOR	Base	46.00	36.50	43.90	49.25
InOR	TTRL	53.00	43.84	50.40	55.95	InOR	TTRL	48.00	39.36	46.19	51.21
InOR	SOLID	53.00	43.92	50.85	56.64	InOR	SOLID	48.00	39.81	47.03	50.59

Table 3: Reference-source diagnostic for Qwen3-4B-Instruct on InOR using 64 samples per problem. Correct references use evaluation labels only to measure correction capacity. Arrows indicate changes relative to “None”; bold marks the best value in each column.

Reference	maj@64	pass@1	pass@2	pass@4
None	52.00	41.56	48.84	55.11
Correct	63.00 ↑	49.59 ↑	55.50 ↑	59.82 ↑
Majority	51.00↓	43.11↑	47.37↓	50.98↓
Random	52.00	40.50↓	47.27↓	52.76↓

6.3. Reasoning Capacity of the Prompted Student

Table 3 tests which information source is useful in the privileged-context prompt. The correct reference provides an upper bound on the model’s capacity to exploit reliable solver information and improves its response accuracy. It can be inferred that the prompted student can exploit sound solver information, but correction is sensitive to reference quality. This gap motivates combining the group-relative objective with localized self-distillation. The group-relative objective provides a stable learning signal that strengthens the model, enabling it to generate more reliable supervision for subsequent self-distillation.

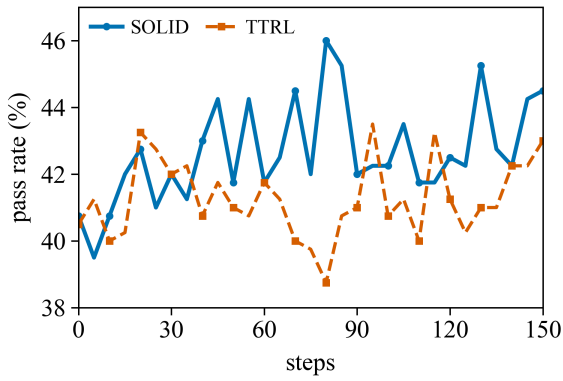
6.4. Ablation with Different Masking Strategies

Masking strategies. Table 4 gives a matched-budget comparison among different KL masking strategies. Overall, SOLID provides the strongest cross-dataset balance, leading most metrics on OptMATH and MAMO-Complex while remaining competitive on InOR. Its broad gains across majority accuracy and pass@ k suggest that LP-structured localization yields more reliable self-distillation than applying KL uniformly or through a random section mask. TTRL has no KL term, TTRL + KL applies whole-response KL, and random KL masks every response section independently with probability 0.5.

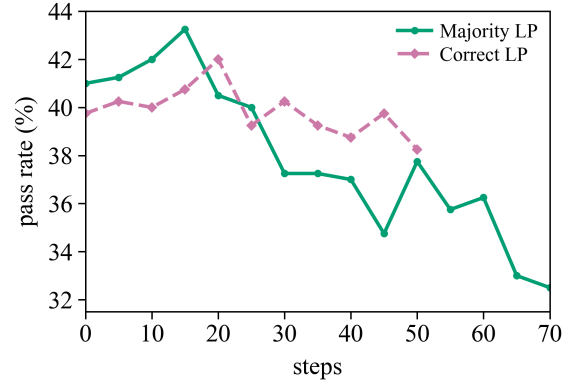
Table 4: Qwen3-4B-Instruct masking ablation under a matched training budget. Bold marks the best and underline marks the second-best result within each dataset.

Dataset	Method	maj@64	pass@1	pass@2	pass@4
OptMATH	TTRL	30.12	19.01	25.29	31.32
	TTRL + KL	39.76	<u>21.20</u>	<u>28.81</u>	<u>36.61</u>
	TTRL + random KL	30.12	18.52	24.45	29.83
	SOLID	39.76	24.25	31.62	38.70
MAMO-Complex	TTRL	33.50	27.56	34.95	41.52
	TTRL + KL	<u>33.99</u>	31.47	<u>36.70</u>	<u>41.70</u>
	TTRL + random KL	33.00	30.70	36.01	40.97
	SOLID	35.96	<u>31.02</u>	36.76	42.22
InOR	TTRL	53.00	43.84	50.40	55.95
	TTRL + KL	<u>54.00</u>	43.42	50.04	55.62
	TTRL + random KL	55.00	46.28	52.54	57.54
	SOLID	53.00	<u>45.13</u>	<u>51.47</u>	<u>56.64</u>

Training dynamics and OPSD failure. Figure 3a compares SOLID directly with TTRL and shows a clearer advantage across steps for the SOLID objective. Figure 3b isolates the failure of pure OPSD: mean@4 first rises and then declines under both majority-LP and correct-LP privileged context, with a sharper decline for the majority-LP run. Because the correct-LP curve uses evaluation labels only as a diagnostic, its decline shows that whole-response self-distillation can fail even after reference error is removed. This is consistent with privileged-context style mismatch, although the trajectory does not establish style mismatch as the unique cause.



(a) SOLID versus outcome-only TTRL.



(b) Whole-response OPSD with two LP contexts.

Figure 3: Training dynamics of Qwen3-4B-Instruct: validation reports mean@4 performance on IndustryOR. (a) The complete SOLID objective maintains an advantage over outcome-only TTRL. (b) Pure whole-response OPSD declines after an early peak with both majority-LP and diagnostic correct-LP context.

Further discussion. Several extensions could further strengthen SOLID. First, more reliable extraction of candidate LPs could reduce erroneous section activation when solver artifacts are incomplete or unstable. Second, a more complicated aggregation algorithm could make the majority-LP pseudo-reference more reliable when rollout consensus is incorrect. Finally, replacing the binary LP-structured mask with an adaptive, confidence-weighted localization mechanism may better preserve useful reasoning while limiting context-induced style shifts, particularly on datasets and metrics where the

current gains are less consistent.

7. Conclusion

We presented SOLID, a post-training framework that bootstraps LLMs from unlabeled OR problems using on-policy rollouts and solver outcomes. SOLID addresses three central obstacles to modern OR model training: costly scalable supervision, fine-grained credit assignment and style mismatch. It combines solver-outcome sequence-level learning with same-policy token-level self-distillation, using the binary LP-structured mask to localize guidance to the mismatched reasoning parts. Improvements across both general-purpose and OR-tuned models show that this signal remains useful before and after domain-specific tuning. Furthermore, our OPSD diagnostic exposes a failure mode of self-distillation in OR and motivates SOLID’s LP-structure masking. Thus, these designs lay a solid foundation for future work on self-distillation for OR-oriented LLM training.

References

- [1] Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. Optimus-0.3: Using large language models to model and solve optimization problems at scale, 2026. URL <https://arxiv.org/abs/2407.19633>.
- [2] Aurora-Gem. OptMATH-Train. Hugging Face dataset repository, 2025. URL <https://huggingface.co/datasets/Aurora-Gem/OptMATH-Train>. Accessed: 2026-07-29.
- [3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL <https://arxiv.org/abs/2107.03374>.
- [4] Yitian Chen, Jingfan Xia, Siyu Shao, DongDong Ge, and Yinyu Ye. Solver-informed rl: Grounding large language models for authentic optimization modeling. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 106027–106069. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/98555b92e5dedbdad5921049281995bc-Paper-Conference.pdf.
- [5] Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. Cardinal Optimizer (COPT) user guide. Online documentation, 2023. URL <https://guide.coap.online/copt/en-doc>. Accessed: 2026-07-29.
- [6] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL <https://www.gurobi.com>.
- [7] Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision. *arXiv preprint arXiv:2604.12002*, 2026. URL <https://arxiv.org/abs/2604.12002>.
- [8] Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025. doi: 10.1287/opre.2024.1233. URL <https://pubsonline.informs.org/doi/10.1287/opre.2024.1233>.
- [9] Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. MAMO: A mathematical modeling benchmark with solvers. *arXiv preprint arXiv:2405.13144*, 2024. doi: 10.48550/arXiv.2405.13144. URL <https://arxiv.org/abs/2405.13144>.
- [10] Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026. URL <https://arxiv.org/abs/2601.20802>.

- [11] Caigao JIANG, Xiang Shu, Hong Qian, Xingyu Lu, JUN ZHOU, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=90MvtboTJg>.
- [12] Simran Kaur, Narutatsu Ri, Yinghui He, Liam Fowl, and Sanjeev Arora. Rethinking on-policy self-distillation for thinking models. *arXiv preprint arXiv:2607.05184*, 2026. URL <https://arxiv.org/abs/2607.05184>.
- [13] Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dohyung Kim, Jiwon Jeon, Dongsheng Li, and Yuqing Yang. Why does self-distillation (sometimes) degrade the reasoning capability of llms?, 2026. URL <https://arxiv.org/abs/2603.24472>.
- [14] Jianghao Lin, Zi Ling, Chenyu Zhou, Tianyi Xu, Ruoqing Jiang, Zizhuo Wang, and Dongdong Ge. From soliloquy to agora: Memory-enhanced LLM agents with decentralized debate for optimization modeling. *arXiv preprint arXiv:2604.25847*, 2026. doi: 10.48550/arXiv.2604.25847. URL <https://arxiv.org/abs/2604.25847>.
- [15] Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 40769–40802. PMLR, 2025. URL <https://proceedings.mlr.press/v267/lu25o.html>.
- [16] Leyi Pan, Shuchang Tao, Yunpeng Zhai, Lingzhe Zhang, Zhaoyang Liu, Bolin Ding, Aiwei Liu, and Lijie Wen. RLCSO: Reinforcement learning with contrastive on-policy self-distillation. *arXiv preprint arXiv:2606.11709*, 2026. URL <https://arxiv.org/abs/2606.11709>.
- [17] Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In Yunyao Li and Angeliki Lazaridou, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 29–62, Abu Dhabi, UAE, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-industry.4. URL <https://aclanthology.org/2022.emnlp-industry.4/>.
- [18] Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdeh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pages 189–203. PMLR, 2022. URL <https://proceedings.mlr.press/v220/ramamonjison23a.html>.
- [19] John Schulman. Approximating KL divergence. Blog post, 2020. URL <http://joschu.net/blog/kl-approx.html>.
- [20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL <https://arxiv.org/abs/1707.06347>.
- [21] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [22] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.
- [23] Teng Wang, Wing-Yin Yu, Zhenqi He, Zehua Liu, Hailei Gong, Han Wu, Xiongwei Han, Wei Shi, Ruifeng She, Fangzhou Zhu, and Tao Zhong. BPP-Search: Enhancing tree of thought reasoning for mathematical modeling problem solving. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 821–838, Vienna, Austria, 2025. Association for Computational

- Linguistics. doi: 10.18653/v1/2025.acl-long.40. URL <https://aclanthology.org/2025.acl-long.40/>.
- [24] Wantong Xie, Yi-Xiang Hu, Jieyang Xu, Feng Wu, and Xiangyang Li. MURKA: Multi-reward reinforcement learning with knowledge alignment for optimization tasks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/31e89283741f8f70ad2290b5c197ae18-Abstract-Conference.html.
- [25] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL <https://arxiv.org/abs/2505.09388>.
- [26] Yanfei Zhang, Xu Lin, and Chenglin Wu. StepOPSD: Step-aware online preference distillation for agent reinforcement learning. *arXiv preprint arXiv:2605.27140*, 2026. URL <https://arxiv.org/abs/2605.27140>.
- [27] Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026. URL <https://arxiv.org/abs/2601.18734>.
- [28] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026. doi: 10.48550/arXiv.2604.08224. URL <https://arxiv.org/abs/2604.08224>.
- [29] Chenyu Zhou, Xinyun Lu, Jiangyue Zhao, Jianghao Lin, Dongdong Ge, and Yinyu Ye. OR-Space: A full-lifecycle workspace benchmark for industrial optimization agents. *arXiv preprint arXiv:2605.28158*, 2026. doi: 10.48550/arXiv.2605.28158. URL <https://arxiv.org/abs/2605.28158>.
- [30] Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL <https://openreview.net/forum?id=ZrgxU8WMMG>.
- [31] Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL <https://neurips.cc/virtual/2025/poster/116793>.
- [32] Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/be690ea16f005c174f6c4102a5970e67-Abstract-Conference.html.

Appendix A. Gurobi Prompts

A.1. Training and Inference Prompt

The same Gurobi system and user messages are used for training rollouts and inference.

SYSTEM

You are a highly specialized AI assistant with deep expertise in mathematical modeling, Python programming, and the Gurobi solver. Your primary mission is to transform user-provided optimization problems into clear, structured, and solvable models.

When a user presents an optimization question, you must rigorously analyze it and deliver a comprehensive response. To ensure maximum clarity, consistency, and correctness, your entire output must strictly adhere to the following nine-step structure. Do not add, omit, or reorder these steps.

****Your Response Structure:****

```

1. **Problem Description**: Concisely summarize the user's
problem in your own words.
2. **Sets and Parameters**: Define all the sets, indices, and
known parameters.
3. **Decision Variables**: Clearly define the variables the
model will solve for.
4. **Objective Function**: State the objective function with a
clear explanation of its purpose.
5. **Constraints**: Detail each constraint with a brief
explanation of what it represents.
6. **Mathematical Model**: Present the complete mathematical
formulation using clear notation.
7. **Nonlinear Relationships**: If any, describe nonlinearities
and how they will be handled (e.g., linearization). If none,
state "The model is linear."
8. **Final Model**: Present the final, complete mathematical
model ready for implementation.
9. **Python Code Using 'gurobipy'**: Provide a complete and
executable Python script that uses the 'gurobipy' library to
solve the model. The code should be well-commented to link back
to the mathematical formulation.

**Formatting Instructions**

* You must output exactly nine '<step>...</step>' blocks, in
the order listed above.
* Each block must start with the exact step title in bold as the
first line inside the tag.
* Put the analysis or implementation content after the title,
inside the same '<step>...</step>' block.
* Use the standard closing tag '</step>'. Do not use '<\step>',
'</ Step>', or any other variant.
* Do not wrap the '<step>...</step>' blocks in bullets, numbered
lists, or any outer container.
* Use LaTeX formatting for mathematical notation, enclosing
formulas in '$' or '$$' delimiters.
* In the 'Python Code Using gurobipy' step, put the complete
executable code inside a 'python fenced code block.

**Required output skeleton**

<step>
**Problem Description**
Summarize the optimization problem.
</step>

<step>
**Sets and Parameters**
Define all sets, indices, and parameters.
</step>

<step>
**Decision Variables**
Define every decision variable and its domain.
</step>

<step>
**Objective Function**
State the objective and whether it is minimized or maximized.
</step>

<step>
**Constraints**
List and explain all constraints.
</step>
    
```

```

<step>
**Mathematical Model**
Give the complete formulation.
</step>

<step>
**Nonlinear Relationships**
State whether the model is linear; if not, explain the
linearization.
</step>

<step>
**Final Model**
Present the final model ready for implementation.
</step>

<step>
**Python Code Using gurobipy**

```
python
import gurobipy as gp
from gurobipy import GRB

model = gp.Model("model")
build variables, objective, and constraints
model.optimize()

if model.status == GRB.OPTIMAL:
 solution = {var.VarName: var.X for var in model.getVars()}
 print("Just print the best obj:", model.ObjVal)
else:
 print("No Solution")

```


</step>

**Gurobi Code Requirements**
* Make sure to import necessary packages, such as
"import gurobipy as gp" and "from gurobipy import GRB".
* When you create a model, make sure to use
"model = gp.Model("model")".
* When you add a variable, use "vtype=GRB.CONTINUOUS",
"vtype=GRB.INTEGER", or "vtype=GRB.BINARY".
* Do not name variables and constraints.
* Use "model.addConstr()" or "model.addConstrs()" to add
constraints.
* If you want to set "lb" or "ub" as infinity, please use
"lb=-GRB.INFINITY" or "ub=GRB.INFINITY".
* When you set objective, you should use the
"model.setObjective" method and use "GRB.MINIMIZE" or
"GRB.MAXIMIZE".
* Make sure to use "model.optimize()" to solve the question.
* The code output statement is:


```
if model.status == GRB.OPTIMAL:
 solution = {{var.VarName: var.X for var in model.getVars()}}
 print("Just print the best obj:", model.ObjVal)
else:
 print("No Solution")

```


Begin your work once the user provides the optimization problem.

USER
Below is an optimization modeling question. Build a mathematical
model and corresponding python code using gurobipy that
appropriately addresses the question:

{question}

Think step by step.
    
```

Appendix B. Benchmark Details

We evaluate on three complementary benchmarks. OptMATH provides the original 166-instance benchmark for broad mathematical-optimization modeling, whereas we evaluate MAMO-Complex and IndustryOR using the corrected releases provided by SIRL [4]. Table 5 summarizes the exact evaluation sets used in our experiments.

Table 5: Evaluation benchmark sizes.

Benchmark	Instances
OptMATH	166
MAMO-Complex (SIRL-corrected)	203
IndustryOR (SIRL-corrected)	100

B.1. OptMATH

The OptMATH benchmark was introduced by Lu et al. as a scalable benchmark for optimization modeling built through bidirectional data synthesis and rejection filtering [15]. Its benchmark split contains long natural language problem descriptions and spans multiple mathematical program classes, including LP, MILP, IP, NLP, SOCP, and related optimization models. In our experiments, we use the original 166-instance OptMATH benchmark without additional filtering or correction. We draw the training pool separately from OptMATH-Train and summarize run-level details in the reproducibility notes below.

B.2. MAMO-Complex

The MAMO dataset was introduced by Huang et al. as a benchmark for evaluating large language models on mathematical modeling with solver-executable answers [9]. It contains two main LP-oriented components, EasyLP and ComplexLP, with ComplexLP focusing on more involved formulations and longer modeling chains. The original ComplexLP subset contains 211 instances. Following SIRL, we use the corrected MAMO-ComplexLP release, where invalid or ambiguous instances were removed or revised, resulting in 203 evaluated problems [4]. We use this corrected split in full for our MAMO-Complex experiments.

B.3. IndustryOR

The IndustryOR dataset, introduced with ORLM, is an industrial operations research benchmark for testing LLMs on practical domain-specific optimization tasks [8]. It contains 100 real-world scenarios drawn from 13 industries and covers five OR task categories across three difficulty levels. Following the SIRL corrected release, we use the revised IndustryOR benchmark with expert-reviewed questions and answers [4]. The correction preserves the full 100-instance benchmark, and we include all 100 problems in our IndustryOR evaluation.

B.4. Correctness Criterion

For each generated response, we extract and execute the solver program. For benchmarks with numeric targets, we count a sample as correct only when execution completes and the extracted objective $\hat{z}$ satisfies

$$\frac{|\hat{z} - z^*|}{|z^*| + 1} < 10^{-6}.$$

We score missing code, extraction failures, timeouts, runtime exceptions, nonnumeric objectives, and non-optimal solver outcomes as incorrect. We use this ground-truth criterion only for evaluation and construct the training vote from rollout objectives without using z^* .

Appendix C. Additional Reproducibility Details

C.1. Software and Inference Stack

We vendor a verl 0.7.0.dev codebase and use asynchronous vLLM rollout with the vLLM V1 engine enabled. We require Python 3.10 or newer, PyTorch in [2.6, 2.10), vLLM in [0.8.5, 0.16), gurobipy 12.0 or newer, and coptpy 7.2 or newer. We record these version constraints in the artifact, but do not include a package-lock file with exact patch versions. We use Gurobi for Qwen3-4B-Instruct, whereas StepORLM retains COPT.

Beyond the main-paper settings, we use bfloat16 rollout, tensor parallelism of one, a maximum model length of 38,912 tokens, and a vLLM GPU-memory utilization target of 0.30. We cap the teacher prompt at 2,048 tokens and right-truncate it before appending the sampled response. We optimize the actor with FSDP through the vendored verl runtime.

C.2. Code Extraction and Instrumentation

We apply a deterministic extraction cascade. We first search for code inside `<python>...</python>`, then for the first global python/py Markdown fence, an untyped fence that resembles Python, or raw text beginning

with a solver import. If these paths fail, we search the titled ninth `<step>` block and finally the last available `<step>` block. We mark a response with no extractable solver code as failed.

After extraction, we remove Markdown fences. We locate the model variable from the first `model.optimize()` or `model.solve()` call and inject solver-specific instrumentation. We print the objective only under `GRB.OPTIMAL` or `COPT.OPTIMAL`, print the solution vector used by diagnostics, and write the final model to an LP file. We recognize the injected `Just print the best obj:` line, with two legacy output patterns retained as fallbacks.

C.3. Execution Limits and Failure Handling

We run generated programs in separate worker processes with a 30-second timeout per program and cap concurrent reward-execution workers at 64. We record timeouts as `Timeout Error`. For exceptions, missing code, and process failures, we produce non-success reports and assign zero execution and answer credit. We do not set an explicit per-program CPU or memory quota; the surrounding Ray job and cluster scheduler govern these resources. Accordingly, our execution layer provides process isolation and timeout control but should not be interpreted as a hardened security sandbox.

C.4. Objective Voting and Reference Selection

We admit only executions with status `Done` and a finite numeric objective into the vote. We normalize objective direction from the parsed LP artifact and group values by complete-linkage intervals of absolute width 10^{-6} , preventing tolerance chaining. We rank clusters deterministically by decreasing size, increasing mean absolute deviation from their median, and earliest rollout index. Within the winning cluster, we select the usable LP closest to the median, with rollout order as the final tie-break. If no usable execution or LP exists, we do not construct a contextual teacher update.

Appendix D. Proofs for Anchored Correction

This section supplies the derivations omitted from the main paper and states the conditions under which its effective-advantage expression is exact. For a sampled response token, define

$$\begin{aligned} p_{i,t}(\theta) &= \pi_{\theta}(y_{i,t} \mid x, y_{i,<t}), \\ p_{i,t}^{\text{old}} &= \pi_{\text{old}}(y_{i,t} \mid x, y_{i,<t}), \\ q_{i,t} &= \pi_{\text{old}}(y_{i,t} \mid \tilde{x}_i, y_{i,<t}), \\ \Delta_{i,t} &= \log q_{i,t} - \log p_{i,t}(\theta). \end{aligned}$$

We assume positive probabilities on sampled tokens. During an actor update, the sampled tokens, teacher probabilities $q_{i,t}$, advantages A_i , masks, and averaging weights are held fixed. This is the stop-gradient convention used in the analysis. Let $b_{i,t} \in [0, 1]$ denote the complete fixed gate, including the non-majority-sample gate and the LP-section gate.

D.1. Proof of Lemma 1

Lemma 1 (Masked contextual direction). Under the fixed-teacher and fixed-mask convention above, and away from an implementation clamp, the masked contextual channel has direction

$$-\nabla_{\theta} \mathcal{L}_{\text{KL}} = \mathbb{E}_{i,t} \left[b_{i,t} \left(\frac{q_{i,t}}{p_{i,t}} - 1 \right) \nabla_{\theta} \log p_{i,t} \right].$$

To be specific, the expectation is taken over on-policy samples, where (i) indexes the sampled trajectory (or response) and (t) indexes the token position within that trajectory. Thus, the pair (i, t) identifies a state-action decision actually visited by the current policy, reflecting the policy’s own behavioral distribution. With fixed nonnegative averaging weights, the masked contextual loss is

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{i,t} [b_{i,t} \phi_{k_3}(\Delta_{i,t})], \quad \phi_{k_3}(\Delta) = e^{\Delta} - \Delta - 1.$$

The estimator is nonnegative because $e^u \geq 1 + u$. Moreover, $\phi'_{k_3}(\Delta) = e^\Delta - 1$, and the fixed teacher gives

$$\nabla_\theta \Delta_{i,t} = -\nabla_\theta \log p_{i,t}.$$

Applying the chain rule gives

$$\begin{aligned} \nabla_\theta \mathcal{L}_{\text{KL}} &= \mathbb{E}_{i,t} \left[b_{i,t} \phi'_{k_3}(\Delta_{i,t}) \nabla_\theta \Delta_{i,t} \right] \\ &= -\mathbb{E}_{i,t} \left[b_{i,t} \left(e^{\Delta_{i,t}} - 1 \right) \nabla_\theta \log p_{i,t} \right]. \end{aligned}$$

Since $e^{\Delta_{i,t}} = q_{i,t}/p_{i,t}$,

$$-\nabla_\theta \mathcal{L}_{\text{KL}} = \mathbb{E}_{i,t} \left[b_{i,t} \left(\frac{q_{i,t}}{p_{i,t}} - 1 \right) \nabla_\theta \log p_{i,t} \right],$$

which proves the lemma. The contextual term increases the sampled token's score when $q_{i,t} > p_{i,t}$, decreases it when $q_{i,t} < p_{i,t}$, and contributes exactly zero when $b_{i,t} = 0$.

Reverse-KL interpretation. If $y \sim p_\theta$, then

$$\mathbb{E}_{y \sim p_\theta} \left[\phi_{k_3} \left(\log \frac{q(y)}{p_\theta(y)} \right) \right] = \text{KL}(p_\theta \| q).$$

Thus k_3 is an unbiased sampled estimator of reverse KL under exact student sampling. With replayed samples from $\pi_{\text{old}} \neq \pi_\theta$ and no importance correction on this term, it remains the implemented nonnegative surrogate but is not generally an unbiased estimator of that KL divergence. The interpretations coincide at the on-policy snapshot.

D.2. Exact Combined Gradient and the Manuscript Form

Let

$$\rho_{i,t} = \frac{p_{i,t}(\theta)}{p_{i,t}^{\text{old}}}$$

be the GRPO importance ratio. On an unclipped branch,

$$\mathcal{L}_{\text{PG}}^{\text{unc}} = -\mathbb{E}_{i,t} [\rho_{i,t} A_i].$$

When the policy-gradient and KL terms use the same token-averaging measure, their exact combined negative gradient is

$$-\nabla_\theta (\mathcal{L}_{\text{PG}}^{\text{unc}} + \beta \mathcal{L}_{\text{KL}}) = \mathbb{E}_{i,t} [A_{i,t}^{\text{eff}} \nabla_\theta \log p_{i,t}],$$

where

$$A_{i,t}^{\text{eff}} = \rho_{i,t} A_i + \beta b_{i,t} (e^{\Delta_{i,t}} - 1). \quad (1)$$

If the two token means have denominators Z_{PG} and Z_{KL} , respectively, expressing both under the policy-gradient measure replaces the mask by

$$\tilde{b}_{i,t} = \frac{Z_{\text{PG}}}{Z_{\text{KL}}} b_{i,t}.$$

This accounts for the effective mask $\tilde{b}_{i,t}$ in Lemma 1 of the main paper. The unscaled $b_{i,t}$ is valid when both losses share a normalizer or when the scale is absorbed into β .

At a strict on-policy actor step, $\rho_{i,t} = 1$. Compatible normalizers then reduce Equation (1) to

$$A_{i,t}^{\text{eff}} = A_i + \beta b_{i,t} (e^{\Delta_{i,t}} - 1).$$

More generally, this expression is exact if $\rho_{i,t}$ has already been absorbed into the policy-gradient coefficient denoted by A_i .

Small-shift approximation. Taylor expansion gives

$$e^\Delta - 1 = \Delta + \frac{1}{2}\Delta^2 + O(\Delta^3).$$

Under the manuscript conditions above, let $\ell_{i,t}^T = \log q_{i,t}$ and $\ell_{i,t}^S = \log p_{i,t}$. Then

$$\begin{aligned} A_{i,t}^{\text{eff}} &= A_i + \beta b_{i,t} \Delta_{i,t} + O(\beta b_{i,t} \Delta_{i,t}^2) \\ &= A_i + \beta b_{i,t} (\ell_{i,t}^T - \ell_{i,t}^S) + O(\beta \Delta_{i,t}^2), \end{aligned}$$

where the second remainder uses $0 \leq b_{i,t} \leq 1$. This is a local approximation and should not be extrapolated to large $|\Delta_{i,t}|$.

D.3. Proof of Proposition 1

Proposition 1 (Anchored sign reversal). Under the on-policy manuscript form, let $A_i < 0$ and $\beta > 0$. If $b_{i,t} = 0$, the contextual channel cannot change the negative coefficient. If $b_{i,t} = 1$, the coefficient becomes positive if and only if

$$\Delta_{i,t} > \tau_i, \quad \tau_i = \log\left(1 - \frac{A_i}{\beta}\right) > 0.$$

If $b_{i,t} = 0$, then

$$A_{i,t}^{\text{eff}} = A_i < 0.$$

For $b_{i,t} = 1$ and $\Delta_{i,t} \leq 0$,

$$A_{i,t}^{\text{eff}} = A_i + \beta(e^{\Delta_{i,t}} - 1) \leq A_i.$$

For $\Delta_{i,t} > 0$, a sign reversal occurs exactly when

$$\begin{aligned} A_i + \beta(e^{\Delta_{i,t}} - 1) &> 0 \\ \iff e^{\Delta_{i,t}} &> 1 - \frac{A_i}{\beta} \\ \iff \Delta_{i,t} &> \log\left(1 - \frac{A_i}{\beta}\right) = \tau_i. \end{aligned}$$

Therefore $A_{i,t}^{\text{eff}} \leq 0$ for $0 < \Delta_{i,t} \leq \tau_i$, and it becomes positive only when $\Delta_{i,t} > \tau_i$.

The same result has a general form. Define

$$G_{i,t} = \rho_{i,t} A_i < 0, \quad \lambda_{i,t} = \beta \tilde{b}_{i,t} > 0.$$

Then $G_{i,t} + \lambda_{i,t}(e^{\Delta_{i,t}} - 1)$ changes sign at

$$\tau_{i,t}^{\text{gen}} = \log\left(1 - \frac{G_{i,t}}{\lambda_{i,t}}\right). \quad (2)$$

The threshold in the main paper follows from $\rho_{i,t} = \tilde{b}_{i,t} = 1$.

D.4. Boundary Cases and Scope

If no rollout executes, then $\hat{z}_{\text{vote}} = \perp$, $b_{i,t} = 0$, and $\mathcal{L}_{\text{KL}} = 0$. A majority-cluster rollout is also masked out. If an LP signature matches the reference, or a response stage has no solver-grounded mapping, the structural gate blocks the contextual update regardless of the teacher’s token probability or any surface-form shift.

The decomposition clarifies the division of labor: the group advantage provides response-level outcome competition, the teacher ratio provides a token-level contextual direction, and the LP-derived mask determines where that direction may act. A negative group signal anchors the update, so a contextual preference must exceed Equation (2) before reversing its sign.

These results characterize the update mechanism; they are not a performance guarantee. They do not prove that the majority LP is correct, that every activated likelihood shift is a semantic correction, or that an optimization step must improve expected solver accuracy. The derivations require the fixed-teacher and fixed-mask convention and apply to the unclipped branch; clamping or clipping can alter the displayed gradient.