Solver-Informed Reinforcement Learning

Yitian Chen, jingfan Xia, Siyu Shao, Dongdong Ge, Yinyu Ye

2025.12

Solver-Informed Reinforcement Learning

Large Reasoning Models (LRMs) are AI models with extensive parameters and advanced logical reasoning capabilities.

Beyond Human-Level Performance
Mathematical proofs · Code generation · STEM competitions

High-Value Application Scenarios
Medical diagnosis · Industrial optimization · Scientific research · Complex decision systems

Reasoning model for optimization modeling

We introduce the first reasoning model for mathematical modeling: **SIRL-Qwen2.5-7B.**

1. **Reasoning Paradigm**: Reasoning → Modeling → Code generation.
2. **Data Synthesis**: Seed data + a precisely designed synthesis framework for model distillation, enabling access to high-quality reinforcement learning training data.
3. **Training Framework**: **Reinforcement Learning with Verifiable Reward (RLVR)--** propose a novel surrogate function design that bridging GRPO(Deepseek) and DAPO(ByteDance).

Reasoning Paradigm: System Prompt

The system prompt guides the LLM through a sequential process: first asking it to analyze the problem; then to formulate a mathematical model; and finally to output the corresponding solver codes.

System Prompt

SYSTEM: You are a helpful Assistant with expertise in operations research and the COPT solver. When the User provides an OR question, you will analyze it, build a detailed mathematical model, and provide the COPT code to solve it.
Your response should follow these steps:
1. <think> Carefully analyze the problem to identify decision variables, objective, and constraints.</think>
2. <model> Develop a complete mathematical model, explicitly defining: * Sets * Parameters * Decision Variables (and their types) * Objective Function * Constraints </model>
3. <python> Provide the corresponding COPT Python code to implement the model. </python>
The output must be in Markdown format, with each step enclosed in the specified tags.
USER: Answer the following mathematical modeling question:
{{Question}}
Please think step by step.

Reasoning Paradigm: Generated Response

杉数科技 Cardinal Operations

Tire Transportation Linear Programming Problem

Thinking

To solve this problem, we need to clarify the decision variables, objective function and constraint conditions:

1. Decision Variables:

- Let x be the number of airplane transportation trips
- Let y be the number of truck transportation trips

2. Objective Function:

We need to minimize the total transportation trips: $x + y$

3. Constraint Conditions:

- The total quantity of tire transportation must reach at least 200 pieces: $10x + 6y \geq 200$
- Total transportation cost cannot exceed $22000: $1000x + 700y \leq 22000$
- Airplane transportation trips cannot exceed truck transportation trips: $x \leq y$
- Non-negativity constraints: $x \geq 0$, $y \geq 0$

This problem conforms to linear constraint characteristics, we can use linear programming method to solve it.

Think

Model

Code

Modeling

Sets

- This problem is a simple linear programming problem, therefore no need to define sets.

Model Parameters

- Single airplane transportation quantity: 10
- Single airplane transportation cost: $1000
- Single truck transportation quantity: 6
- Single truck transportation cost: $700

Decision Variables

- x: Number of airplane transportation trips
- y: Number of truck transportation trips

Objective Function

Minimize total transportation trips: $x + y$

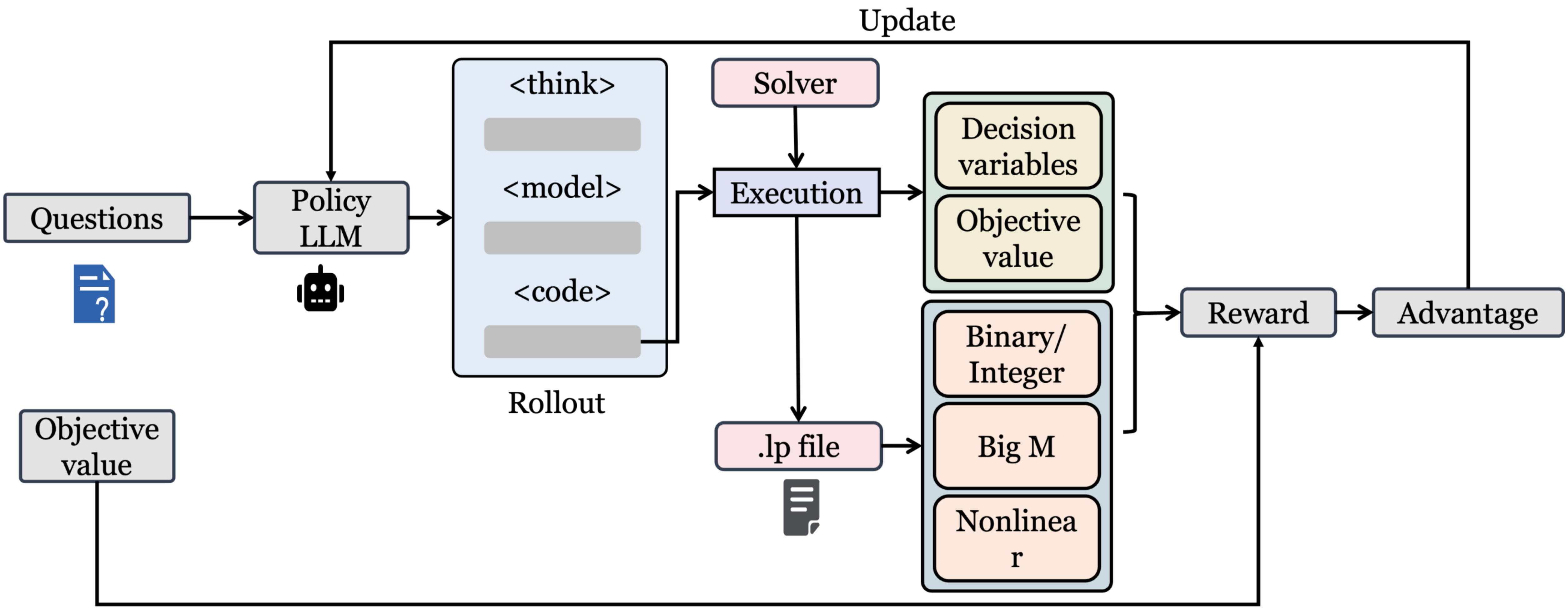
Constraint Conditions

1. $10x + 6y \geq 200$
2. $1000x + 700y \leq 22000$
3. $x \leq y$
4. $x \geq 0$, $y \geq 0$

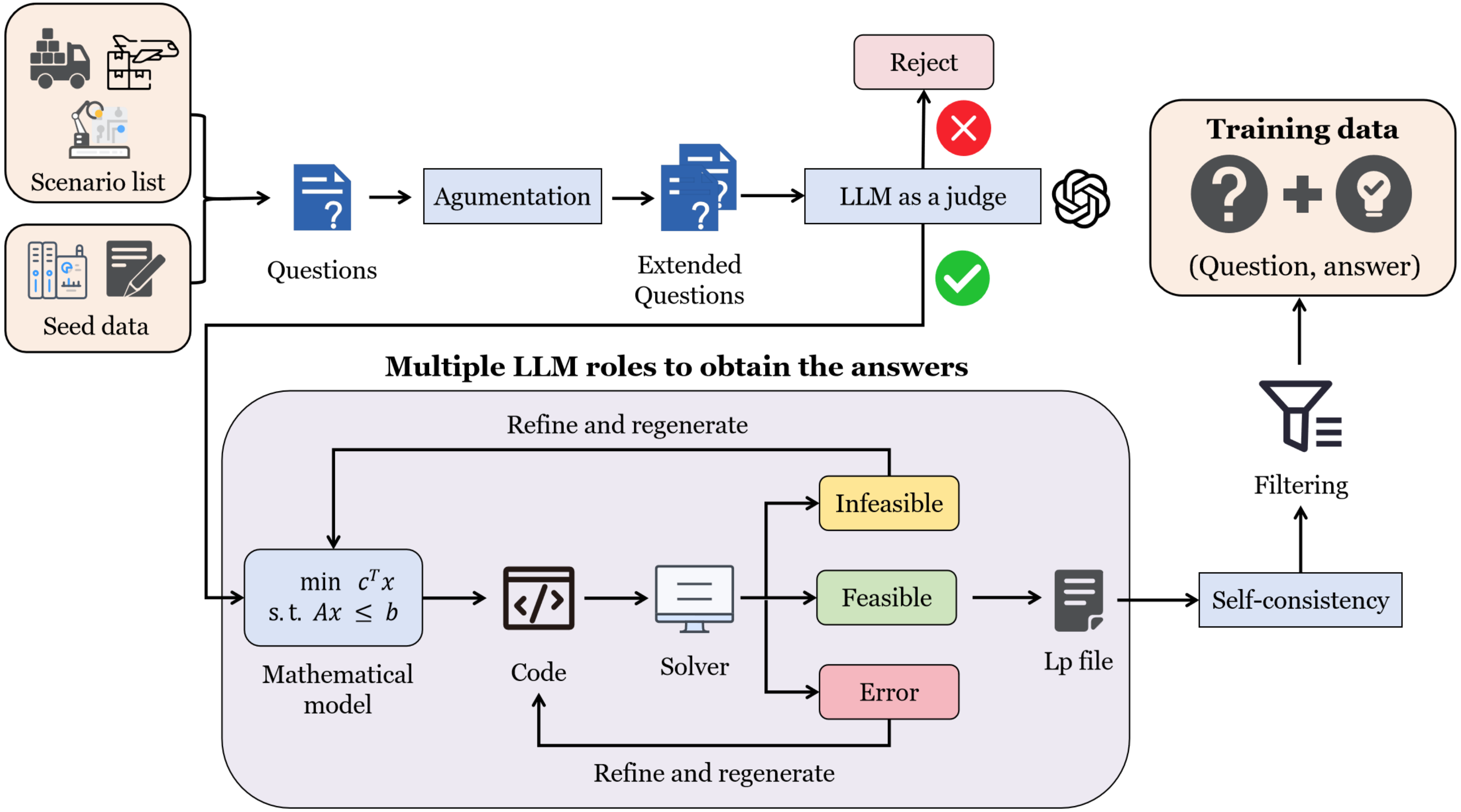
Python Code Implementation

```python
import coptpy as cp
from coptpy import COPT

### Create new model
config = cp.Envr()
env = cp.Envr(config)
m = env.createModel("tire_transport")

### Decision variable definition
x = m.addVar(name="x", vtype=COPT.INTEGER)  # Number of airplane transportation trips
y = m.addVar(name="y", vtype=COPT.INTEGER)  # Number of truck transportation trips

### Objective function setting
m.setObjective(x + y, COPT.MINIMIZE)

### Constraint condition construction
m.addConstr(10 * x + 6 * y >= 200, "tire_constraint")
m.addConstr(1000 * x + 700 * y <= 22000, "cost_constraint")
m.addConstr(x - y <= 0, "trip_constraint")

### Model solving
m.solve()
### Result output
if m.status == COPT.OPTIMAL:
    print(f"Optimal solution found:")
    print(f"Plane trips: {x.x}")
    print(f"Truck trips: {y.x}")
    print(f"Total trips: {x.x + y.x}")
    print(f"Total cost: ${m.objVal}")
else:
    print("No optimal solution found.")
```

SIRL: RL training framework overall

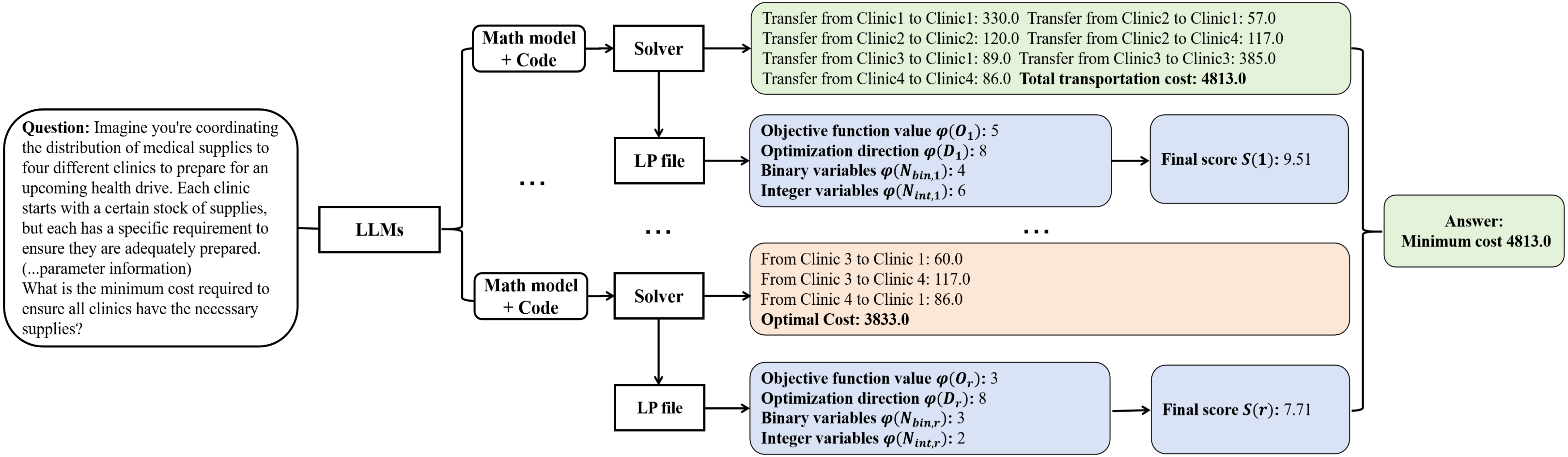


Data Synthesis: Overall Framework



- "LLM as a judge" validates the generated problems.
- An iterative reflection and refinement process is employed to address execution issues.
- Multiple LLM roles (10 roles) per problem for self-consistency.

Data Synthesis: Instance-Enhanced Self-Consistency



- **Instance-Enhanced Self-Consistency (I-ESC):** Incorporates structural metadata from generated LP files (e.g., objective value, direction, binary/integer variable counts) to enforce consensus.
- **Complexity Expansion:** Systematically enhances the dataset's **coverage of complex and challenging problems**.

Surrogate Function Design: Partial KL

杉数科技 Cardinal Operations

Three distinct surrogate function designs:

1. **Full KL:** the standard approach applying full KL-divergence regularization against the reference policy: PPO, Reinforce++;
2. **Without KL:** an approach omitting KL-divergence regularization, which is popular in RLVR training for mathematical problems: DAPO;
3. **Partial KL:** our novel design that applies the KL penalty selectively to the mathematical formulation and code segments.

Partial KL employs selective KL regularization, serving a dual purpose:

1. **Exploration:** KL regularization is omitted for early reasoning steps ($z^1, \ldots, z^{m-2}$), promoting exploration and the identification of diverse problem structures.
2. **Stability:** For critical modeling z^{m-1} and code generation z^m segments, KL regularization ensures well-structured output and prevents policy collapse, facilitating stable, reward-driven improvement.

Surrogate Function Design: Partial KL

Reasoning paradigm: the system prompt guides the reasoning response generation into m distinct segments:

- $(\mathbf{z}^1, \ldots, \mathbf{z}^{m-2})$: Initial reasoning and problem analysis segments.
- $\mathbf{z}^{m-1}$: modeling formulation segment.
- $\mathbf{z}^{m}$: executable codes segment.

The final output y is generated by executing the code segment z^m using the deterministic execution function g, resulting in $y = g(x, z)$.
Partial KL surrogate function design: selectively applies the KL penalty to the mathematical formulation $\mathbf{z}^{m-1}$ and solver code $\mathbf{z}^m$ segments. The value for the KL term, $\mathrm{KL}(j, t)$, within these segments is computed using the unbiased estimator described in [17]:

$$\mathrm{KL}(j,t) = \begin{cases} \frac{\pi_\theta(\mathbf{z}_t|\mathbf{x},\mathbf{z}^{<j})}{\pi_{\theta_{\text{old}}}(\mathbf{z}_t|\mathbf{x},\mathbf{z}^{<j})} - \log \frac{\pi_\theta(\mathbf{z}_t|\mathbf{x},\mathbf{z}^{<j})}{\pi_{\theta_{\text{old}}}(\mathbf{z}_t|\mathbf{x},\mathbf{z}^{<j})} - 1 & j \in \{m-1, m\}, \\ 0 & \text{otherwise.} \end{cases}$$

Reward Design: Two-Stage, Rule-Based Mechanism

杉数科技 Cardinal Operations

the two-stage reward function $r(x, \mathbf{z}, y^*)$ is defined as follows:

$$r(x, \mathbf{z}, y^*) = \begin{cases} R_{\text{format}}(\mathbf{z}) + R_{\text{exec}}(\mathbf{z}) + R_{\text{accur}}(x, \mathbf{z}, y^*) & \text{Stage-1}, \\ R_{\text{format}}(\mathbf{z}) + R_{\text{exec}}(\mathbf{z}) + R_{\text{accur}}(x, \mathbf{z}, y^*) + R_{\text{bonus}}(x, \mathbf{z}, y^*) & \text{Stage-2}. \end{cases}$$

1. **Stage-1** focuses on building fundamental skills for standard optimization problem formulation and solving.

2. **Stage-2** aims to address more complex problems by using a bonus reward R_{bonus} based on the generated mathematical model to encourage advanced modeling techniques (e.g., Big-M, nonlinear).

Main Results

杉数科技
Cardinal Operations

Table: Performance comparison of models on benchmarks.

Types	Models	Acc (pass@1) NL4OPT	MAMO Easy	MAMO Complex	IndustryOR	OptMATH	Macro AVG
Baseline	GPT-4	89.0%*	87.3%*	49.3%*	33.0%*	16.6%*	55.0%*
	DeepSeek-V3.1	84.8%	88.9%	63.5%	44.0%	43.9%	65.0%
LRMs	DeepSeek-R1	82.4%	87.2%	**67.9%**	**45.0%**	40.4%	64.6%
	OpenAI-o3	69.4%	77.1%	51.2%	44.0%	44.0%	57.1%
Agent-based	OptiMUS	78.8%*	77.2%*	43.6%*	31.0%*	20.2%*	49.4%*
Offline-learning	ORLM-LLaMA-3-8B	85.7%*	82.3%*	37.4%*	24.0%*	2.6%*	46.4%
	LLMOpt-Qwen2.5-14B	80.3%*	89.5%*	44.1%*	29.0%*	12.5%*	51.1%
	OptMATH-Qwen2.5-7B	94.7%*	86.5%*	51.2%*	20.0%*	24.4%*	55.4%
	OptMATH-Qwen2.5-32B	95.9%*	89.9%*	54.1%*	31.0%*	34.7%*	61.1%
Online-RL	SIRL-Qwen2.5-7B	96.3%	91.7%	51.7%	33.0%	30.5%	60.6%
	SIRL-Qwen2.5-32B	**98.0%**	**94.6%**	61.1%	42.0%	**45.8%**	**68.3%**

Values marked with * are from original or reproduced papers with the criterion: relative error $< 10^{-6}$.

1. **Our SIRL-7B Our SIRL-7B model consistently and significantly outperforms all other 7B and 14B offline learning models.**
2. **Furthermore, our 32B model surpasses the Macro Average of much larger models, including the 671B Deepseek-V3.1 and leading reasoning models like DeepSeek-R1 and OpenAI-o3.**

Surrogate Function Design: Ablation Study

Table: Ablation study on different surrogate function designs.

Type	MAMO Complex		IndustryOR		OptMATH	
	Acc(pass@1)	ER	Acc(pass@1)	ER	Acc(pass@1)	ER
Partial KL	51.7%	98.1%	33.0%	96.0%	30.5%	92.2%
Full KL	48.3%(↓3.4%)	98.5%(↑0.4%)	30%(↓3.0%)	95.0%(↓1.0%)	28.3%(↓2.2%)	93.4%(↑1.2%)
Without KL	47.3%(↓4.4%)	95.6%(↓2.5%)	29%(↓4.0%)	87.0%(↓9.0%)	29.5%(↓1.0%)	80.1%(↓12.1%)

This table reports both **pass@1** accuracy and **execution rate (ER)**, which measures the percentage of generated solutions that successfully compile and return a valid result.

Full KL: the standard approach applying full KL-divergence regularization against the reference policy;

Without KL: an approach omitting KL-divergence regularization, which is popular in RLVR training for mathematical problems such as AIME;

Partial KL: our novel design that applies the KL penalty selectively to the mathematical formulation and code segments.

Case Study: Execution Error Under Without KL Strategy

杉数科技 Cardinal Operations

1. Omitting the KL divergence in the code component leads to **execution errors**.

2. The response from the Without KL model attempts to enforce that decision variables are greater than zero, but the **implemented code is non-functional** since '>' not supported between instances of 'Var' and 'int'.

Case Study 1: Question instance from the IndustryOR dataset

A product can be processed on any one of the four devices: A, B, C, or D. The preparation completion costs when each device is enabled, the unit production cost for the product, and the maximum processing capacity of each device are shown in Table 5-7. If 2000 units of the product need to be produced, how can the total cost be minimized? Try to establish a mathematical model.
Table 5-7 Device,Prep Completion Cost (Yuan),Unit Production Cost (Yuan/Unit),Maximum Processing Capacity (Units) A,1000,20,900 B,920,24,1000 C,800,16,1200 D,700,28,1600

Case Study 1: Generated response under the `Without KL` strategy

```
<python>
import coptpy as cp
from coptpy import COPT
# Define the model
config = cp.Envr()
env = cp.Envr(config)
model = env.createModel("Product_Optimization")
# Parameters
completion_cost = {"A": 1000, "B": 920, "C": 800, "D": 700}
production_cost = {"A": 20, "B": 24, "C": 16, "D": 28}
max_processing_capacity = {"A": 900, "B": 1000, "C": 1200, "D": 1600}
# Decision variables
x = {}
for i in completion_cost.keys():
x[i] = model.addVar(name=i, lb=0, ub=max_processing_capacity[i], vtype=COPT.CONTINUOUS)
# Objective function
model.setObjective(cp.quicksum(completion_cost[i] * (x[i]>0) + production_cost[i] * x[i] for i in completion_cost.keys()), COPT.MINIMIZE)
# Constraints
model.addConstr(cp.quicksum(x[i] for i in completion_cost.keys()) == 2000, "Total_Production")
# Optimize the model
model.solve()
# Print the solution
if model.status == COPT.OPTIMAL:
print("Optimal Solution:")
for i in completion_cost.keys():
print(f"Device i: {x[i].x}")
print(f"Total Cost: model.objVal")
else:
print("No optimal solution found.")
</python>
```

Two-Stage Reward Mechanism: Ablation Study

杉数科技 Cardinal Operations

Table: Performance results of the ablation study on reward design.

Reward Type	Acc (pass@1)				
	NL4OPT	MAMO Easy	MAMO Complex	IndustryOR	OptMATH
Two-stage rewards	96.3%	91.7%	51.7%	33.0%	30.5%
Stage-1 reward only	96.7% (↑0.4%)	88.8% (↓2.9%)	46.8% (↓4.9%)	27.0% (↓6.0%)	28.9% (↓1.6%)
Stage-2 reward only	92.2% (↓4.1%)	89.6% (↓2.1%)	49.3% (↓2.4%)	28.0% (↓5.0%)	33.1% (↑2.6%)

1. **Stage-1 reward** yielded strong performance on NL4OPT, indicating effective learning of fundamental optimization skills.
2. While **stage-2 reward** optimized OptMATH via advanced strategies, it negatively impacted simpler NL4OPT performance.
3. The **combined two-stage reward** successfully balanced learning objectives, outperforming single-stage rewards across most tasks by resolving inherent trade-offs.

SIRL: Summary

1. **Contribution/Novelty**: We introduce the first domain-specific reasoning model for optimization modeling, establishing the initial application of RLVR (Reinforcement Learning with Variable Reasoning) for LLMs in this domain.

2. **Performance:** Our 32B model achieves a higher Macro Average than much larger models, surpassing the 671B Deepseek-V3.1 and leading reasoning models (e.g., DeepSeek-R1, OpenAI-o3).

3. **Technical Innovation :** We propose a Partial KL-based surrogate function design for LLMs in optimization modeling, significantly boosting both confidence and accuracy across optimization tasks.

Github	https://github.com/Cardinal-Operations/SIRL
Huggingface	https://huggingface.co/chenyitian-shanshu/SIRL
Modelscope	https://modelscope.cn/models/oneday88/SIRL-7B

Q&A

杉数科技
Cardinal Operations

THANKS

400-680-5680 www.shanshu.ai shanshu@shanshu.ai