OptMATH: A Scalable Bidirectional Data Synthesis Framework for Optimization Modeling

Zhonglin Xie

Beijing International Center for Mathematical Research
Peking University

Joint work with Hongliang Lu, Yaoyu Wu, Can Ren, Yuxuan Chen, Zaiwen Wen

Accepted as a poster presentation at ICML 2025

June 28, 2025

Outline

Background

The challenge, and art, in using convex optimization is in **recognizing and formulating the problem**. *Once this formulation is done, solving the problem is, like least-squares or linear programming, (almost) technology.*[1]

Description

A company has three transportation options to choose from to transport 25 tons of cargo, namely trucks, airplanes, and ships with costs $100, $120, $80 per ton and capacities of 10, 20, 30 tons respectively. The company can' t choose trucks and ships together. How should the company optimize the selection and allocation of these methods to minimize overall costs?

Formulation

Variables :

x_1, x_2, x_3 0-1variables indicating whether trucks, airplanes, and ships are are selected, respectively.

y_1, y_2, y_3 Non-negative continuous variables indicating the volume of cargo.

Objectives:

Minimize $100y_1 + 120y_2 + 80y_3$

Constraints:

$$x_1 + x_2 + x_3 \geq 1$$

$$y_1 \leq 10x_1, y_2 \leq 20x_2, y_3 \leq 30x_3$$

$$x_1 + x_3 \leq 1$$

$$y_1 + y_2 + y_3 \geq 25$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

Python Code

```
import gurobipy as gp
from gurobipy import GRB

# Create Model
model = gp.Model("Cargo_Transportation")

# Define decision variables
y1 = model.addVar(vtype=GRB.CONTINUOUS,
name="Trucks_Tons", lb=0)
……………
# Objectives
model.setObjective(100*y1 + 120*y2 + 80*y3,
GRB.MINIMIZE)
# Constraints
model.addConstr(y1 + y2 + y3 >= 25,
"Total_Cargo")
……………
# Optimize
model.optimize()
# Print the result
if model.status == GRB.OPTIMAL:
……………
```

[1] Boyd, Stephen. "Convex optimization." Cambridge UP (2004).

LLMs for Automated Optimization Modeling

Motivation: Although solver technologies are quite advanced, the process of building optimization models still heavily relies on human expertise. The goal of automated modeling is to reduce this dependency, allowing more people without expertise in optimization to benefit from optimization techniques.

Related Works:

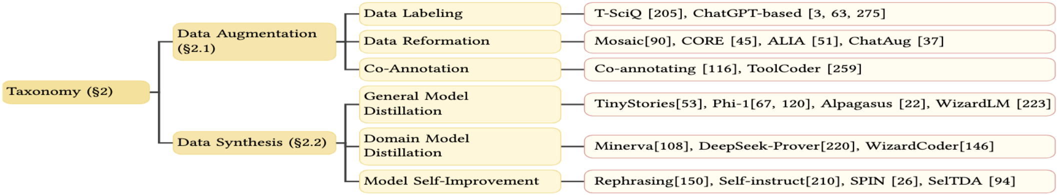
- **NL4Opt Competition** - Initial exploration of using LLMs for assisted modeling.
- **OptiMUS** - Prompt engineering and agent-based approach.
- **ORLM** - Synthetic data and model fine-tuning approach.
- **LLMOPT** - Model fine-tuning and alignment approach.

Main Challenges:

- Prompt-based methods rely on LLMs' inherent modeling capability without enhancing it.
- Learning-based methods lack large-scale, high-quality optimization problem datasets.

Data Synthesis

- Fine-tuning relies heavily on training data and the selection of a base model.
- The release of o1 has sparked significant interest in data synthesis.
- Existing data synthesis methods fall into two categories [2]:



  - **Data Augmentation**: Enhances existing samples through augmentations techniques.
  - **Data Synthesis**: Creates new samples from scratch or via GPT.

Challenges: How to synthesize large-scale, high-quality data for Optimization Modeling?

[2] Wang, Ke, et al. "A survey on data synthesis and augmentation for large language models." arXiv preprint arXiv:2410.12896 (2024).

Outline

Standard Optimization Problem

Standard Form:

$$\begin{aligned} \min_{\mathbf{x}} \quad & g(\mathbf{x}), \\ \text{subject to} \quad & c_i(\mathbf{x}) = 0, \quad i \in \mathcal{E}, \\ & c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

Where:

- $\mathbf{x} \in \mathbb{R}^n$: Decision vector.
- $g : \mathbb{R}^n \rightarrow \mathbb{R}$: Objective function.
- $c_i : \mathbb{R}^n \rightarrow \mathbb{R}$: Constraint functions.
- $\mathcal{E}, \mathcal{I}$: Index sets for equality and inequality constraints respectively.

Main Challenges:

- Modern solvers (e.g., Gurobi, Mosek) can efficiently solve optimization problems using algorithms like interior-point methods.
- The primary challenge lies in transforming real-world problems into precise mathematical formulations.

Problem Formulation

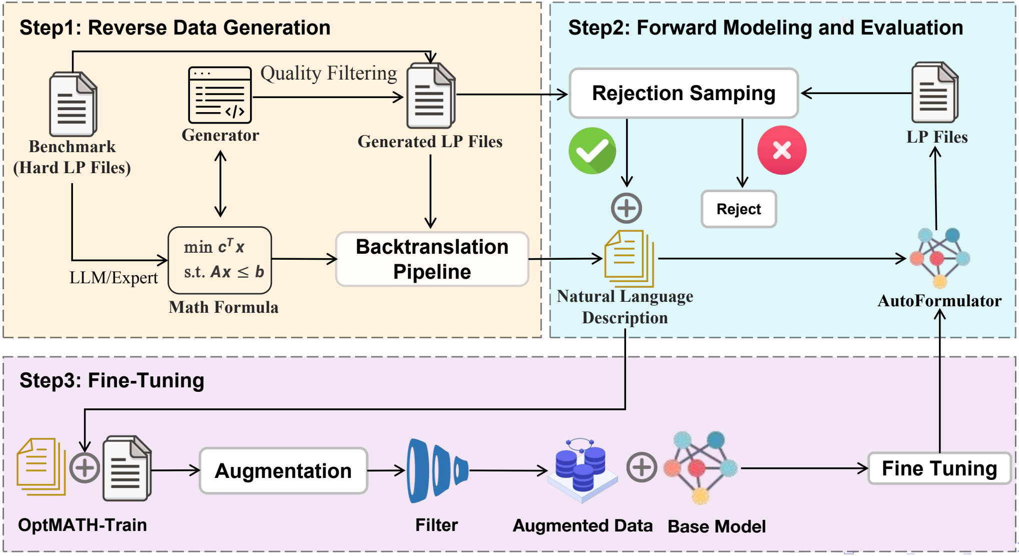
The formulation for increasing the modeling capability of the LLM can be expressed as:

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{(\mathrm{NL},\mathrm{MF},\mathrm{PD})\sim\mathcal{D}}[Q_{(\mathrm{NL},\mathrm{MF},\mathrm{PD})}(\mathrm{MF}', \mathrm{PD}')] \\ \text{s.t.} \quad & (\mathrm{MF}', \mathrm{PD}') = \mathcal{A}_\theta(\mathtt{prompt}_{\mathrm{M}}(\mathrm{NL})) \end{aligned}$$

Key Components:

- $\mathcal{A}_\theta$: Large Language Model with parameters θ
- Q: Quality metric for evaluation
- $\mathcal{D}$: Distribution of problem instances
- $\mathtt{prompt}_{\mathrm{M}}$: Modeling prompt template
- NL: Natural Language Description
- MF: Mathematical Formulation (abstract)
- PD: Problem Data (concrete, solver-ready)

An Overview of our pipeline



Outline

Seed Problem Classes

To build our training dataset, we started by curating 53 distinct optimization problem generators, enabling scalable generation of diverse problem instances. Here's a simplified example of our bin packing generator:

```
class BinPackingGenerator:
    def __init__(self, n_items=(3,10), weight_range=(1,50),
                 bin_capacity=100, seed=None):
        self.params = locals()
        if seed: random.seed(seed)

    def generate_instance(self):
        n = random.randint(*self.params['n_items'])
        weights = {i: random.randint(*self.params['weight_range'])
                   for i in range(n)}

        model = gp.Model("BinPacking")
        x = model.addVars(n, n, vtype=GRB.BINARY)
        y = model.addVars(n, vtype=GRB.BINARY)

        model.setObjective(y.sum(), GRB.MINIMIZE)
        model.addConstrs((sum(weights[i]*x[i,j]
                     for i in range(n)) <=
                     self.params['bin_capacity']*y[j]
                     for j in range(n)))

        return model
```

Problem Data Generation Algorithm

- To ensure a balanced distribution of problem difficulty, we designed an algorithm as shown in Algorithm 1.
- The core idea is to control problem difficulty using LLM and a complexity score function:

$$
\begin{aligned}
S(\mathrm{PD}) = {} & \alpha_{\text{bin}} N_{\text{bin}} + \alpha_{\text{int}} N_{\text{int}} + \alpha_{\text{cont}} N_{\text{cont}} \\
& + \beta_{\text{lin}} N_{\text{lin}} + \beta_{\text{indic}} N_{\text{indic}} + \beta_{\text{quad}} N_{\text{quad}} \\
& + \beta_{\text{gen}} N_{\text{gen}} + \gamma_{\text{BigM}} f_{\text{BigM}} + \delta_{\text{expr}} \overline{L_{\text{expr}}}
\end{aligned}
$$

Algorithm 1 Feedback-Driven Problem Data Generation

Require: Target complexity range $[S_{\min}, S_{\max}]$, time limits $[T_{\min}, T_{\max}]$, instance generator G, feasibility threshold $\mathcal{F}_{\text{target}}$, max iterations T

Ensure: Configuration Θ such that for $\mathrm{PD}_i \sim G(\Theta)$: $S(\mathrm{PD}_i) \in [S_{\min}, S_{\max}]$ (complexity), $\tau_i \leq T_{\max}$ (solving time), $\Pr(f_i = \texttt{feasible}) \geq \mathcal{F}_{\text{target}}$

Initialize parameters via LLM:
    $\Theta_0 \leftarrow \mathcal{L}(\texttt{prompt}_{\text{IC}}(S_{\min}, S_{\max}, T_{\min}, T_{\max}))$
for $t = 1$ **to** T **do**
    Generate N PDs: $\{\mathrm{PD}_i\}_{i=1}^{N} \leftarrow G(\Theta_{t-1})$
    Compute metrics: $S(\mathrm{PD}_i)$ (Eq. 4), τ_i (solving time), f_i (feasibility)
    Aggregate statistics: $\bar{S}_t = \frac{1}{N}\sum S(\mathrm{PD}_i)$, $\bar{\tau}_t = \frac{1}{N}\sum \tau_i$, $\mathcal{F}_t = \frac{1}{N}\sum \mathbb{I}(f_i = \texttt{feasible})$
    if $\bar{S}_t \in [S_{\min}, S_{\max}]$ **and** $\bar{\tau}_t \leq T_{\max}$ **and** $\mathcal{F}_t \geq \mathcal{F}_{\text{target}}$ **then**
        return Θ_{t-1}
    else
        Refine parameters via feedback:
            $\Theta_t \leftarrow \mathcal{L}(\texttt{prompt}_{\text{RC}}(\bar{S}_t, \bar{\tau}_t, \mathcal{F}_t; \Theta_{t-1}))$
    end if
end for
return $\emptyset$ (no valid Θ found)

Example: Measuring Problem Complexity

Production Planning Problem:

- **Variables:**
  - Binary: $y_1, y_2 \in \{0, 1\}$ (production decisions)
  - Integer: $x_1, x_2 \in \mathbb{Z}^+$ (production quantities)
  - Continuous: $z \geq 0$ (total cost)
- **Objective:** $\min \; z + 10y_1 + 8y_2$

Constraint Types:

1. Linear: $2x_1 + 3x_2 \leq 100$, $x_1 \leq 50$, $x_2 \leq 30$
2. Indicator (Big-M): $x_1 \geq 5 - 100(1 - y_1)$
3. Quadratic: $z \geq 0.5x_1^2 + 0.3x_2^2$
4. Nonlinear: $x_1 e^{x_2} \leq 100$

Complexity Analysis:

- Variable counts:
  - 2 binary
  - 2 integer
  - 1 continuous
- Constraint counts:
  - 3 linear
  - 2 indicator
  - 1 quadratic
  - 1 nonlinear
- Big-M frequency: $f_{\text{BigM}} = 2$
- Avg expr length: $\overline{L_{\text{expr}}} \approx 2.71$
- Score: $S = 16.71$ (unit weights)

Outline

Bidirectional Data Synthesis Algorithm

We design a three-phase backtranslation pipeline to generate high-quality problem descriptions:

- **Initial Generation:** LLM generates initial NL description from mathematical formulation and problem data
- **Self-Criticism:** LLM evaluates the description by examining mathematical equivalence, completeness, and clarity
- **Self-Refinement:** Based on criticism, LLM generates refined descriptions focusing on accuracy and completeness

Algorithm 2 Bidirectional Data Synthesis Algorithm

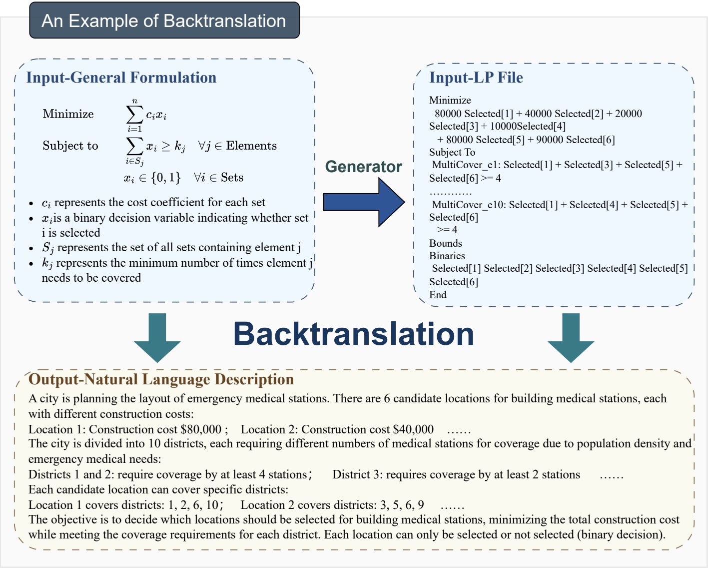
Require: Instance pair $(\mathrm{MF}_i, \mathrm{PD}_{i,j})$, Max Iteration T
Ensure: $(\mathrm{NL}_{i,j}, \mathrm{MF}'_{i,j}, \mathrm{PD}'_{i,j}, \mathrm{OV}_{i,j})$

```
Initial generation: NL ← 𝓛(prompt_I(MF_i, PD_{i,j}))
Initialize: SC = SR = Null
for k = 1, ..., T − 1 do
    Self-Criticize:
    SC ← 𝓛(prompt_C(MF_i, PD_{i,j}, NL))
    Self-Refine:
    SR ← 𝓛(prompt_R(MF_i, PD_{i,j}, NL, SC, SR))
    if SR is good enough then
        break
    end if
end for
NL_{i,j} ← SR
AutoFormulation:
(MF'_{i,j}, PD'_{i,j}) ← 𝒜_θ(prompt_M(NL_{i,j}))
OV_{i,j} ← Solve PD_{i,j} by Gurobi
OV'_{i,j} ← Solve PD'_{i,j} by Gurobi
if OV_{i,j} = OV'_{i,j} then
    return (NL_{i,j}, MF'_{i,j}, PD'_{i,j}, OV_{i,j})
else
    return Null
end if
```

An Example of Backtranslation

- The backtranslation pipeline converts mathematical formulations (MF) and problem data (PD) into human-readable problem descriptions.

- To ensure the correctness and consistency of generated descriptions with respect to MF and PD, we perform rejection sampling on the outputs.



Forward Modeling and Rejection Sampling

Forward Modeling:

- AutoFormulator transforms NL to MF and PD.
- Uses Chain-of-Thought prompting strategies.
- Generates diverse modeling reasoning paths.

Rejection Sampling Process:

- Compare solutions: $\mathrm{OV}'_{i,j}$ (from generated) vs $\mathrm{OV}_{i,j}$ (from original).
- Accept if $\mathrm{OV}_{i,j} = \mathrm{OV}'_{i,j}$.
- Manual validation shows 99.6% accuracy!

CoT Prompt Example

The following is an operations research problem. Let's solve it step by step:

1. Identify the decision variables, objective function, and constraints
2. Formulate the mathematical model
3. Implement the solution using Gurobi in Python
4. Verify and interpret the results

Training Strategy

Data Augmentation

- Multiple augmentation strategies including:
  - Problem rewriting
  - Semantic substitution
  - Constraint expansion
- Dual sampling quality control

Augmentation Prompt Examples

1. Rewrite the problem using different expressions and terminology while keeping the core optimization task identical.
2. Generate a variant by adding/removing/-modifying one constraint while maintaining problem feasibility.

Iterative Training Process

- Parameter-efficient fine-tuning with LoRA
- Joint optimization objective:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(p,y)\sim\mathcal{D}_{\mathrm{SFT}}} \left[\sum_{t=1}^{|y|} \log P_\theta(y_t | y_{<t}, p) \right]$$

Outline

Problem Data Distribution

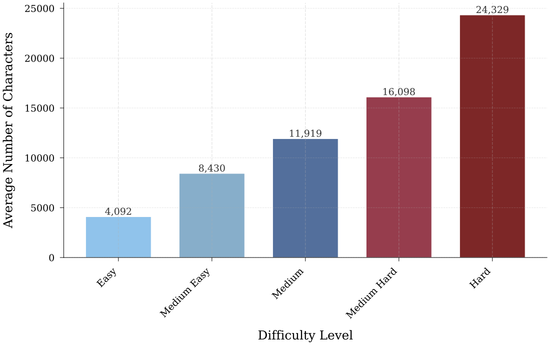


Figure: Distribution of LP file lengths across generated instances by difficulty levels.

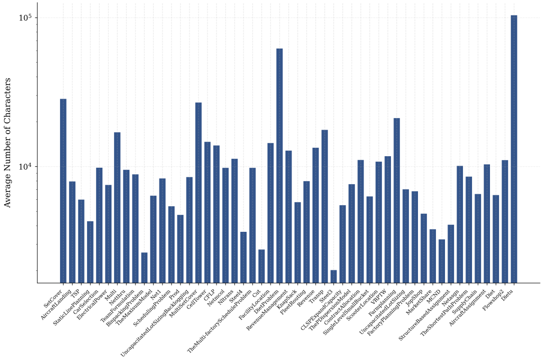


Figure: Distribution of LP file lengths across generated instances by problem types.

Statistics of the OptMATH Dataset

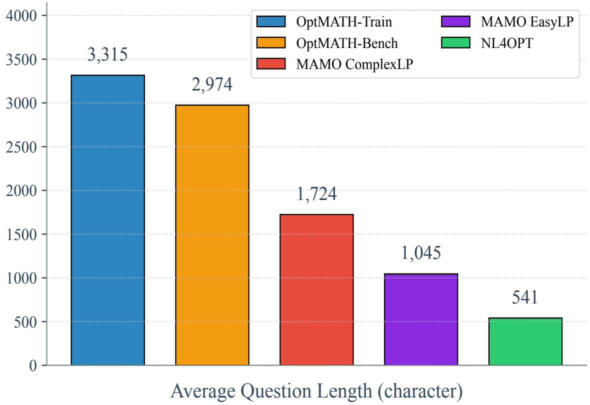
Problem Length Analysis:

- OptMATH presents significantly **more complex problem descriptions** compared to existing benchmarks.

Problem Type Coverage:

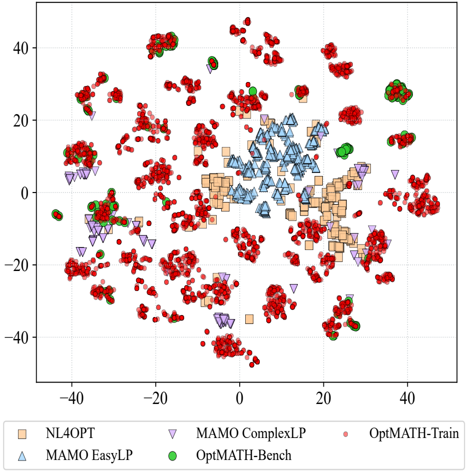
- OptMATH-Bench covers a wide range of optimization problems:
    - Linear Programming
    - Mixed Integer Linear Programming
    - Integer Programming
    - Nonlinear Programming
    - Second-Order Cone Programming

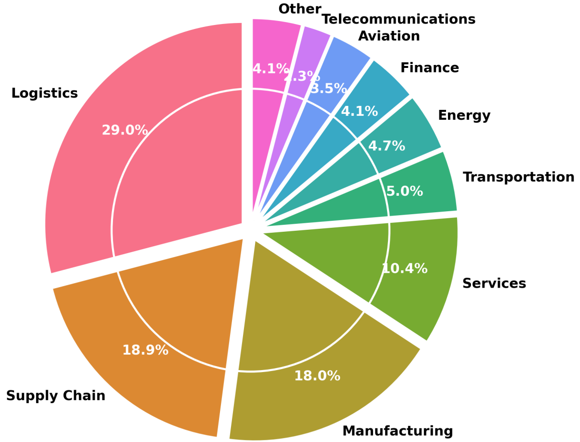
Type	NL4OPT	MAMO EasyLP	MAMO ComplexLP	OptMATH-Bench
IP	26.94%	58.74%	15.64%	11.92%
LP	61.63%	0.61%	62.08%	8.81%
MILP	11.43%	40.65%	22.28%	67.87%
NLP	0.00%	0.00%	0.00%	5.18%
SOCP	0.00%	0.00%	0.00%	6.22%



Statistics of the OptMATH Dataset

OptMATH demonstrates comprehensive coverage across both problem space and application domains, with its embeddings surrounding existing benchmarks while spanning diverse industrial scenarios.





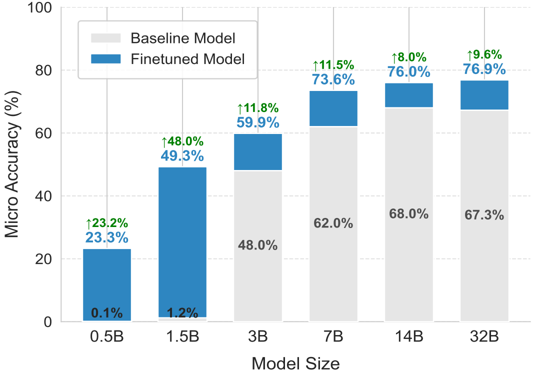
Performance Comparison of Models on Different Benchmarks

Table 1. Performance Comparison of Models on Different Benchmarks

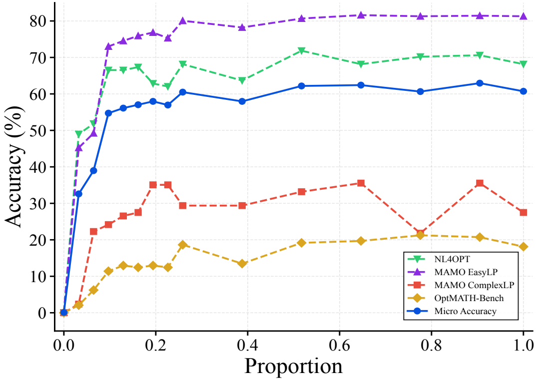
Types	Models	NL4OPT	MAMO EasyLP	MAMO ComplexLP	OptMATH Bench	IndustryOR	OptiBench	Macro AVG
Baseline	Llama3.1-8B	0.0%	0.2%	0.0%	0.0%	0.0%	0.0%	0.1%
	Qwen2.5-7B	86.9%	83.6%	21.8%	1.6%	10.0%	36.2%	40.0%
	GPT-3.5-turbo	78.0%	79.3%	33.2%	15.0%	21.0%	47.4%	51.4%
	GPT-4	89.0%	87.3%	49.3%	16.6%	33.3%	68.6%	57.4%
	Deepseek-V3	95.9%	88.3%	51.1%	32.6%	37.0%	71.6%	62.8%
	OptiMUS (GPT-4o-2024-05-13)	78.8%	77.0%	43.6%	20.2%	31.0%	45.8%	49.4%
	Qwen2.5-32B	92.7%	82.2%	44.6%	9.3%	16.0%	47.6%	48.7%
Fine-tuning	ORLM-Llama-3-8B (reported)	85.7%	82.3%	37.4%	*	38.0%	*	60.9%
	ORLM-Llama-3-8B (reproduced)	84.5%	74.9%	34.1%	2.6%	24.0%	51.1%	45.2%
	OptMATH-Llama3.1-8B (pass@1)	55.5%	73.9%	40.8%	24.4%	18.0%	55.5%	44.7%
	OptMATH-Qwen2.5-7B (pass@1)	94.7%	86.5%	51.2%	24.4%	20.0%	57.9%	55.8%
	OptMATH-Qwen2.5-32B (pass@1)	**95.9%**	**89.9%**	**54.1%**	**34.7%**	**31.0%**	**66.1%**	**62.0%**
Pass@8	OptMATH-Llama3.1-8B	97.6%	94.2%	71.6%	51.6%	37.0%	66.6%	69.8%
	OptMATH-Qwen2.5-7B	98.4%	94.5%	72.5%	56.0%	38.0%	68.1%	71.3%
	OptMATH-Qwen2.5-32B	97.9%	93.9%	75.4%	67.4%	47.0%	76.8%	76.4%

Scaling Analysis on Model Size and Training Data Size

OptMATH demonstrates consistent performance gains across model sizes (0.5B-32B) and training data scales, with larger models achieving better absolute performance while smaller models show higher sensitivity to data scaling.



Model size scaling (0.5B-32B)



Training data scaling (Qwen2.5-1.5B)

Many Thanks For Your Attention!

Homepage: zlxie.cn
arXiv Link: https://arxiv.org/abs/2502.11102