Large Language Models as End-to-end Combinatorial Optimization Solvers

Xia Jiang[1], Yaoxin Wu[1], Minshuo Li[1], Zhiguang Cao[2], Yingqian Zhang[1]

Eindhoven University of Technology, The Netherlands [1]
Singapore Management University, Singapore [2]

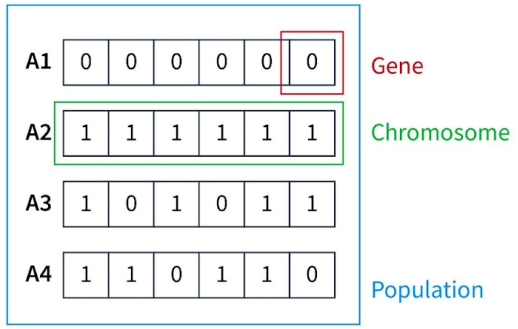
Overview

- Background
- Methodology
- Experiment Results

Background

Combinatorial Optimization (CO)

CO problems are traditionally solved by:



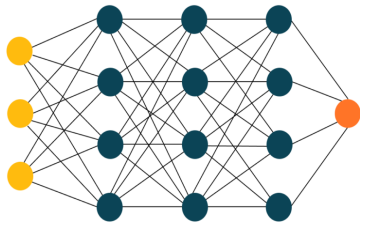
(meta)heuristics

GUROBI OPTIMIZATION

IBM CPLEX

Google OR-Tools

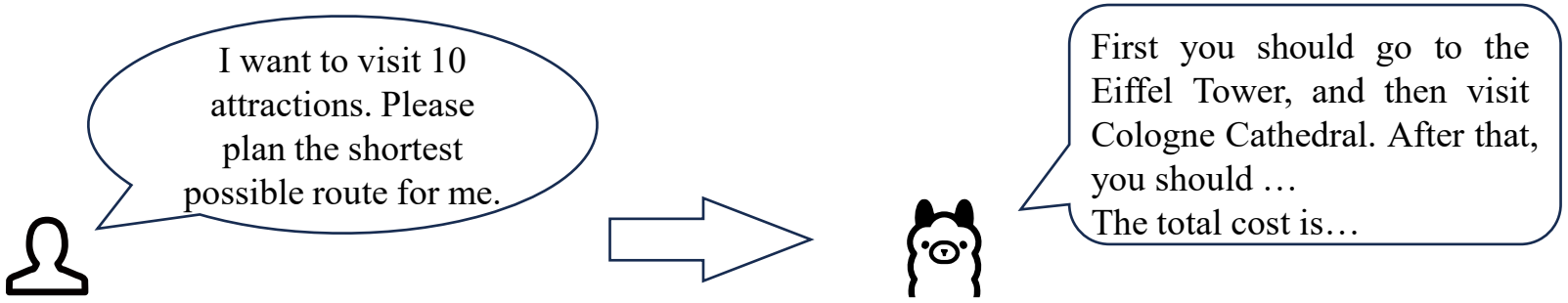
Specialized optimization solvers



Neural networks

For professional users

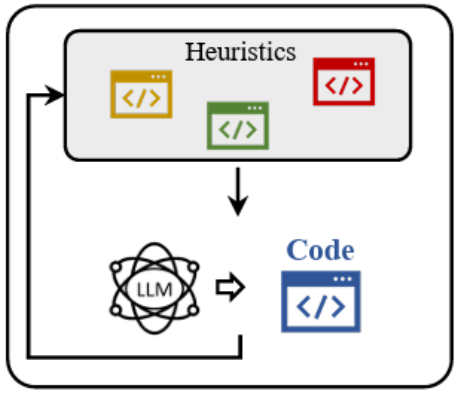
With large language models (LLMs):



We hope that non-expert users can also optimize their lives.

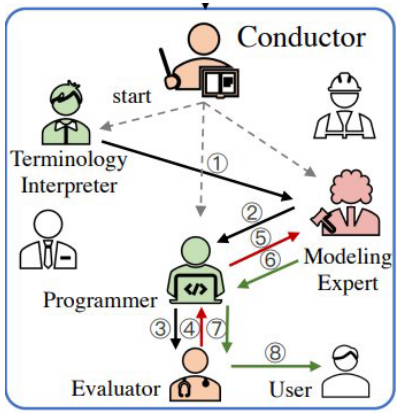
Current LLM-based methods

LLM as heuristic generator



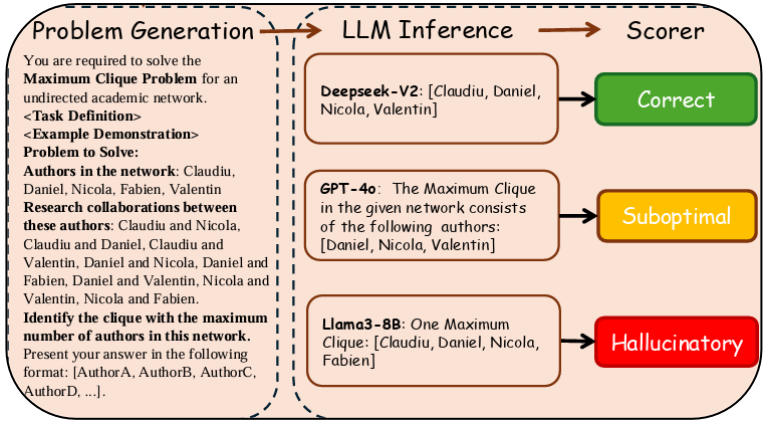
LLMs are used to discover new problem-specific heuristic algorithms, requiring **domain expertise to define and use the algorithms**.

LLM as solver code generator



LLMs are taken as agents to call the specialized optimization solvers, requiring **solver licenses and domain expertise to run the code**.

LLM as optimizer



LLMs are directly prompted to generate solutions in an end-to-end manner. Such methods exhibit **significant performance gaps**.

Image credits:

Liu, Fei, et al. "Evolution of heuristics: Towards efficient automatic algorithm design using large language model." ICML 2024.
Xiao, Ziyang, et al. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. ICLR 2023.
Tang, Jianheng, et al. GraphArena: Evaluating and Exploring Large Language Models on Graph Computation. ICLR 2025.

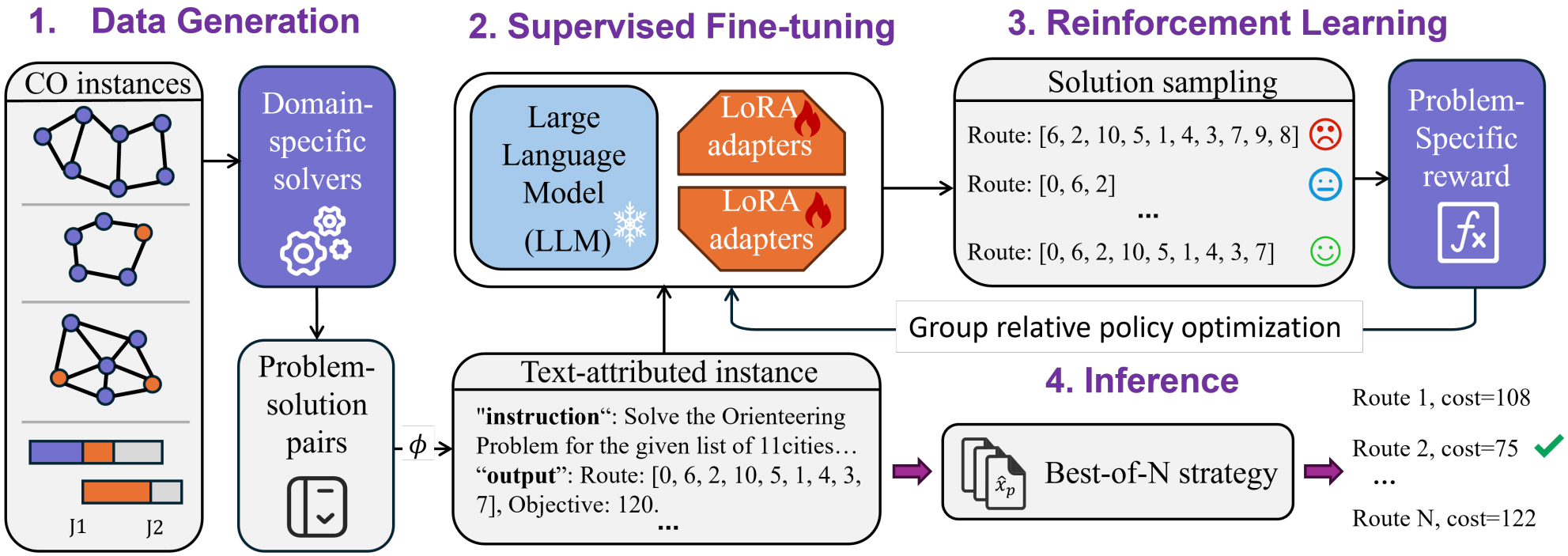
Background

When asking DeepSeek-R1

Okay, I need to solve the TSP for 48 cities using coordinates and two nearest neighbors per city. TSP is NP-hard, so I'll try a heuristic.
Maybe nearest neighbor? But that might cause loops...
Alternatively, maybe I can start building the route by choosing a starting city and then always moving to the nearest unvisited neighbor...
Alternatively, I could use some kind of greedy heuristic that selects the globally shortest available edge that doesn't create a cycle...
Alternatively, perhaps I can use the given neighbor data to form a minimum spanning tree...
Alternatively, maybe I can look at the set of neighbors and try to trace a Hamiltonian cycle manually...
Hmm... Christofides algorithm? No, we lack full distance matrix...
Alternatively, maybe I can use the neighbor data to identify subtours and then link them...
Wait, I'm stuck at Node 16. Maybe start over...
Looks like two clusters based on Y-coordinates. Let's try connecting them. From Node 29 to Node 40 seems shortest ($\approx$363 units)...
Alternatively, maybe I can build a tour per cluster and then stitch them...
Wait, how to close the cycle? Still missing cities... Maybe patch cycles? Or build a spanning tree? Eulerian? No, degrees don't match...
Let's just assume we can approximate the route, sum known neighbor distances, and plug in inter-cluster links. Total distance? No idea.
Maybe ~5000? Actually, forget it — too complex manually.
Alternatively, the problem may have a specific structure where the route can be determined by connecting each node...But as before, this gets stuck. Given the time I've spent and recognizing that a precise manual solution isn't feasible, I'll provide the route as per the nearest neighbor heuristic, acknowledging that it may not visit all nodes but following the given instructions as best as possible.

Deepseek-R1 got stuck multiple times when trying to construct the solution of a TSP instance.
Finally, it acknowledge that the problem is too difficult to be solved effectively.

Methodology



- **Data Generation:** Generate high-quality solutions to CO problems using domain-specific solvers, and transform the problem-solution pairs to natural language for model training.
- **Supervised Fine-tuning**: Enable language models to learn solutions patterns for different CO problems directly from problem-solution pairs through tuning LoRA adapters.
- **Reinforcement learning**: Mitigate constraint violation of the trained policy and improve solution quality.
- **Best-of-N inference:** Enhance model performance through test-time exploration.

Methodology

Supervised fine-tuning

CO is formulated as a **next-token prediction** task, treating it as a **language generation problem** where the LLM learns to map textual problem descriptions to textual solution representations.

Standard language modelling loss is applied

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{i=1}^{n} \log \Pr_\theta(y_i | \phi(p), y_{<i}),$$

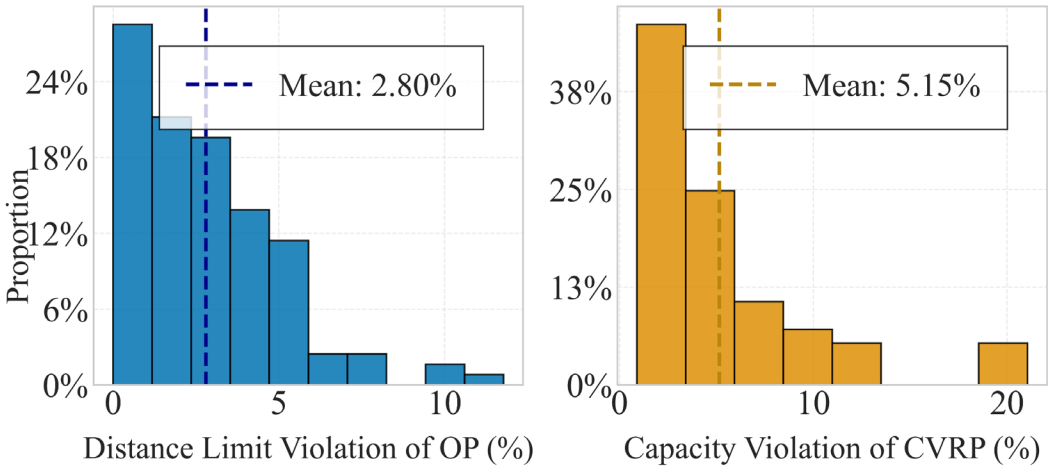
$\phi(p)$ - the input prompt

y_i- the i_{th} token in the textual solution

$y_{<i}$- all previous tokens of the solution

Orienteering problem (OP)

Capacitated vehicle routing problem (CVRP)



SFT policies are over-greedy: they tend to violate problem-specific constraints for some CO problems.

Most violations stem from slight oversteps of constraint:

- traveling marginally farther than the distance limit in OP
- serving slightly more customer demand than the capacity limit in CVRP

Methodology

Rewards for reinforcement learning

- **Feasibility-aware reward:**

$$\mathcal{R}_f^{\mathcal{P}}(\hat{x}_p) = \begin{cases} \omega_0\zeta + \sum_{i=1}^{m_p} \omega_i c_{p,i}(\hat{x}_p) & \text{if } \zeta \neq 0 \\ 0 & \text{if } \zeta = 0 \end{cases}, \forall p \in \mathcal{P}$$

- **Optimality-aware reward:**

$$\mathcal{R}_o^{\mathcal{P}}(\hat{x}_p) = \begin{cases} \alpha \frac{1}{1+M_o(\hat{x}_p)} & \text{if } \zeta \neq 0 \\ 0 & \text{if } \zeta = 0 \end{cases}, \forall p \in \mathcal{P}$$

$\hat{x}_p$- the LLM-generated solution

ω_i- the weights for constraint i

$c_{p,i}$- the i_{th} rule-based constraint

Feasibility-aware rewards (CVRP example)

- The solution can be parsed by the given format: 0.1
- The routes start/end at the depot: 0.2
- The vehicle capacity is not violated: 0.5
- All customers are visited exactly once: 0.2

Group relative policy optimization (GRPO)

$$\mathcal{L}_{\text{FOARL}}(\theta) = \mathbb{E}_{\phi(p)\sim D^{\mathcal{P}}, \{\hat{x}_{p,i}\}_{i=1}^{S}\sim\pi_{\theta,\text{old}}(\hat{X}_p|\phi(p))} \left[\frac{1}{S}\sum_{i=1}^{S} \left(\min\left(r_i^{\text{ratio}} A_i, \text{clip}\left(r_i^{\text{ratio}}, 1-\epsilon, 1+\epsilon \right) A_i \right) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right) \right]$$

$$A_i = (R_i^P - mean(R_{group}^P)) \,/\, std(R_{group}^P)$$

GRPO trains LLMs by generating **multiple responses** for the same prompt, ranking them, and updating the model to **increase the likelihood of better-ranked answers while suppressing worse ones**.

Experiment Results

Settings

Base LLM: Qwen-2.5-7B

SFT data size: 500,000 instances per problem

RL data size: at most 3200 instances

Only the LoRA adapters (2.08% parameters of the LLM) are fine-tuned.

Evaluation metrics

Feasibility rate (fea.): the proportion of feasible solutions relative to the total test data.

Optimality gap (opt.): the percentage difference between LLM-generated solution and the optimal solution.

Baselines

General-purpose LLMs (One-shot prompt): GPT-4o, GPT-4o-mini, Claude-3.7-Sonnet, Claude-3.5-Haiku, DeepSeek-V3-671B, LLaMA-3.3-70B, and Qwen2.5-72B.

Reasoning LLMs: GPT-o3-mini, GPT-o1, and DeepSeek-R1

Prompt Strategies: Optimization by PROmpting (OPRO), LMEA (LLM-driven Evolutionary Algorithm), progressive-hint prompting (PHP), self-guiding exploration(SGE).

Domain-specific heuristics. The widely-used heuristic solvers, such as OR-tools, greedy algorithms, and dispatching rules.

LLM-based model-then-solve methods: LLMOPT [1], DRoC [2], and ORLM [3].

[1] Jiang, Caigao, et al. "LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch." ICLR 2025.
[2] Jiang, Xia, et al. "DRoC: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints." ICLR 2025.
[3] Huang, Chenyu, et al. “ORLM: A customizable framework in training large models for automated optimization modeling." Operations Research (2025).

Experiment Results

Table 1: Evaluation of feasibility (fea.), optimality (opt.), and average time (Avg. Time) for different methods on the 7 studied CO problems. The best results are in bold.

Method	TSP		OP		CVRP		MIS		MVC		PFSP		JSSP		Avg. Time
	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	fea.	opt.	
General-purpose Language Models															
GPT-4o	39%	33.79%$_{\pm 16.6}$	59%	55.19%$_{\pm 15.7}$	15%	76.62%$_{\pm 7.9}$	8%	11.70%$_{\pm 11.8}$	6%	16.67%$_{\pm 7.0}$	88%	20.57%$_{\pm 9.2}$	7%	97.85%$_{\pm 23.7}$	5.3s
GPT-4o-mini	28%	53.38%$_{\pm 21.5}$	80%	70.70%$_{\pm 10.3}$	3%	68.10%$_{\pm 10.5}$	1%	0.00%$_{\pm 0}$	3%	29.52%$_{\pm 9.7}$	78%	21.47%$_{\pm 10.0}$	8%	212%$_{\pm 121}$	4.5s
Claude-Sonnet	66%	24.53%$_{\pm 10.7}$	49%	34.62%$_{\pm 14.1}$	30%	38.34%$_{\pm 15.9}$	13%	12.51%$_{\pm 12.5}$	2%	6.25%$_{\pm 6.3}$	100%	18.42%$_{\pm 8.9}$	10%	90.00%$_{\pm 21.6}$	5.4s
Claude-Haiku	45%	38.60%$_{\pm 16.9}$	26%	51.26%$_{\pm 14.9}$	4%	61.48%$_{\pm 12.1}$	2%	23.33%$_{\pm 6.7}$	6%	43.01%$_{\pm 23.3}$	73%	20.58%$_{\pm 7.6}$	9%	95.51%$_{\pm 22.2}$	5.1s
DeepSeek-V3	73%	35.75%$_{\pm 15.4}$	50%	46.10%$_{\pm 13.4}$	21%	58.22%$_{\pm 26.8}$	5%	12.05%$_{\pm 12.9}$	15%	37.15%$_{\pm 24.8}$	58%	20.81%$_{\pm 9.4}$	52%	103.19%$_{\pm 26.9}$	26.4s
Llama3.3-70B	50%	69.08%$_{\pm 31.4}$	27%	48.98%$_{\pm 14.6}$	31%	97.31%$_{\pm 69.3}$	8%	37.12%$_{\pm 29.5}$	20%	22.86%$_{\pm 13.6}$	98%	21.97%$_{\pm 8.4}$	29%	105.01%$_{\pm 24.5}$	2.1s
Qwen2.5-72B	20%	36.89%$_{\pm 34.6}$	32%	49.36%$_{\pm 16.6}$	61%	180.91%$_{\pm 105}$	14%	29.56%$_{\pm 16.2}$	5%	63.20%$_{\pm 30.4}$	98%	21.13%$_{\pm 8.2}$	53%	103.90%$_{\pm 61.7}$	12.5s
Reasoning Models															
GPT-o3-mini	91%	306%$_{\pm 258}$	8%	43.93%$_{\pm 11.5}$	50%	139%$_{\pm 39.7}$	66%	9.23%$_{\pm 8.4}$	33%	2.98%$_{\pm 5.5}$	98%	16.97%$_{\pm 9.8}$	20%	77.86%$_{\pm 37.8}$	1.4m
GPT-o1	54%	276%$_{\pm 242}$	31%	40.90%$_{\pm 16.3}$	24%	154%$_{\pm 117}$	82%	8.03%$_{\pm 12.9}$	47%	3.58%$_{\pm 5.9}$	89%	14.86%$_{\pm 10.5}$	29%	81.90%$_{\pm 29.6}$	3.2m
DeepSeek-R1	48%	70.99%$_{\pm 23.1}$	60%	40.54%$_{\pm 13.7}$	26%	30.46%$_{\pm 18.1}$	41%	1.60%$_{\pm 3.6}$	38%	4.17%$_{\pm 6.3}$	100%	16.65%$_{\pm 8.1}$	5%	26.29%$_{\pm 8.5}$	6.5m
Prompt Strategies															
OPRO	83%	35.98%$_{\pm 3.6}$	85%	53.96%$_{\pm 14.3}$	21%	37.03%$_{\pm 18.3}$	7%	5.95%$_{\pm 7.3}$	9%	41.67%$_{\pm 16.9}$	99%	18.40%$_{\pm 8.4}$	65%	83.35%$_{\pm 25.5}$	2.1m
LMEA	77%	265%$_{\pm 131}$	48%	66.18%$_{\pm 10.5}$	24%	61.24%$_{\pm 19.0}$	5%	25.0%$_{\pm 14.2}$	13%	34.22%$_{\pm 16.3}$	98%	14.31%$_{\pm 7.1}$	44%	83.19%$_{\pm 23.9}$	5.3m
PHP	84%	33.84%$_{\pm 14.6}$	43%	36.08%$_{\pm 15.1}$	33%	58.11%$_{\pm 26.4}$	5%	11.67%$_{\pm 9.1}$	13%	19.84%$_{\pm 10.01}$	92%	17.23%$_{\pm 8.1}$	56%	104.04%$_{\pm 29.2}$	1.6m
SGE	98%	29.66%$_{\pm 43.3}$	93%	24.49%$_{\pm 38.4}$	84%	36.14%$_{\pm 59.2}$	92%	3.62%$_{\pm 7.6}$	94%	3.83%$_{\pm 7.4}$	95%	4.48%$_{\pm 7.4}$	87%	38.58%$_{\pm 49.2}$	3.6m
Ours															
SFT	89%	2.30%$_{\pm 1.9}$	54%	2.32%$_{\pm 2.6}$	59%	6.02%$_{\pm 3.9}$	80%	1.71%$_{\pm 3.9}$	98%	2.41%$_{\pm 3.3}$	100%	2.22%$_{\pm 1.9}$	100%	11.01%$_{\pm 7.9}$	5.6s
SFT+RL	91%	2.32%$_{\pm 2.2}$	92%	4.25%$_{\pm 2.9}$	80%	8.27%$_{\pm 5.6}$	83%	1.34%$_{\pm 3.3}$	98%	2.39%$_{\pm 3.2}$	100%	2.12%$_{\pm 1.8}$	100%	10.94%$_{\pm 7.3}$	5.6s
SFT+RL+BoN	**100%**	**1.07%**$_{\pm 0.9}$	**100%**	**1.85%**$_{\pm 1.7}$	**100%**	**4.53%**$_{\pm 3.5}$	**94%**	**1.04%**$_{\pm 3.4}$	**100%**	**1.29%**$_{\pm 2.2}$	**100%**	**1.03%**$_{\pm 1.1}$	**100%**	**8.20%**$_{\pm 6.3}$	9.8s

- The current LLMs and prompting strategies exhibit **substantial gaps in solution optimality**.
- The proposed SFT enables the LLM to generate **mostly feasible solutions** with small optimality gaps.
- By further combining SFT with RL and BoN inference, our method achieves a **100% feasibility rate across 6 tasks.**

Experiment Results

	Method	Small graphs Gap	Gap@1	Gap@5	Gap@10	Medium graphs Gap	Gap@1	Gap@5	Gap@10	Large graphs Gap	Gap@1	Gap@5	Gap@10
TSP	OR-Tools	0.82%	76%	96%	99%	2.59%	28%	86%	99%	3.59%	12%	80%	99%
	NN	19.36%	3%	5%	19%	24.13%	0%	0%	1%	26.19%	0%	0%	2%
	FI	2.27%	51%	82%	96%	4.41%	8%	64%	95%	4.86%	2%	53%	99%
	ACO	1.98%	48%	88%	100%	17.98%	0%	1%	6%	36.69%	0%	0%	0%
	Ours	0.14%	96%	100%	100%	0.70%	74%	100%	100%	1.34%	44%	100%	100%
OP	Greedy	16.06%	4%	8%	25%	18.91%	0%	0%	5%	20.30%	0%	0%	3%
	GI	9.07%	20%	37%	63%	11.91%	0%	12%	51%	13.63%	0%	7%	32%
	Tsili	3.85%	21%	68%	96%	9.54%	0%	2%	55%	13.80%	0%	0%	8%
	ACO	3.49%	30%	76%	94%	6.24%	1%	35%	89%	7.95%	0%	15%	74%
	Ours	1.47%	54%	95%	99%	2.04%	26%	96%	100%	2.10%	27%	96%	99%
CVRP	OR-Tools	3.60%	45%	69%	93%	7.87%	3%	24%	72%	8.84%	0%	15%	71%
	Sweep	18.36%	8%	18%	32%	20.59%	0%	1%	8%	22.07%	0%	0%	3%
	PS	3.95%	24%	72%	93%	5.67%	2%	50%	89%	6.12%	0%	41%	89%
	ACO	2.52%	44%	81%	96%	17.07%	0%	2%	14%	29.49%	0%	0%	0%
	Ours	1.70%	52%	90%	97%	4.57%	8%	59%	98%	7.24%	1%	19%	84%
MIS	Degree	5.89%	57%	57%	67%	7.52%	32%	42%	64%	9.61%	20%	33%	59%
	Greedy	2.56%	84%	84%	86%	2.79%	67%	71%	90%	3.36%	53%	60%	83%
	Ours	0.38%	97%	98%	98%	1.05%	86%	87%	94%	2.29%	47%	57%	65%
MVC	Approx	49.79%	0%	0%	0%	39.21%	0%	0%	0%	35.35%	0%	0%	0%
	Greedy	2.80%	71%	72%	89%	2.62%	44%	79%	99%	2.21%	33%	89%	100%
	Degree	3.53%	63%	63%	86%	2.78%	44%	78%	98%	2.63%	26%	83%	99%
	Ours	0.48%	93%	93%	100%	1.25%	66%	94%	100%	2.35%	38%	87%	100%
PFSP	Palmer's	30.52%	0%	0%	1%	30.41%	0%	0%	0%	30.68%	0%	0%	0%
	NEH	1.33%	53%	97%	99%	2.78%	11%	90%	100%	3.56%	0%	88%	100%
	Ours	0.25%	85%	100%	100%	1.16%	40%	100%	100%	2.62%	6%	99%	100%
JSSP	SPT	19.58%	2%	4%	17%	25.32%	0%	0%	1%	27.35%	0%	0%	0%
	FIFO	24.38%	2%	4%	12%	32.97%	0%	0%	1%	39.00%	0%	0%	0%
	ATC	20.71%	0%	12%	15%	24.30%	0%	0%	1%	27.99%	0%	0%	0%
	Ours	2.86%	32%	79%	98%	9.56%	0%	8%	60%	16.25%	0%	0%	4%

Gap@K: the percentage of instances solved with an optimality gap below K%

Our models **outperform the commonly used domain-specific algorithms**, especially on smaller graphs, indicating that solving CO problems described by language is possible.

Method	TSP-small fea.↑	TSP-small opt.↓	TSP-medium fea.↑	TSP-medium opt.↓	TSP-large fea.↑	TSP-large opt.↓
LLMOPT	100%	0.00%	100%	1.00%	100%	7.39%
DRoC	100%	0.00%	100%	0.00%	100%	2.48%
ORLM	100%	0.18%	100%	18.43%	73%	301.00%
Ours	100%	0.14%	100%	0.70%	100%	1.34%
	OP-small		**OP-medium**		**OP-large**	
	fea.↑	**opt.↓**	**fea.↑**	**opt.↓**	**fea.↑**	**opt.↓**
LLMOPT	100%	0.74%	100%	13.91%	100%	66.00%
DRoC	100%	1.04%	100%	9.73%	100%	42.93%
ORLM	100%	2.97%	100%	21.68%	100%	56.16%
Ours	100%	1.47%	100%	2.04%	100%	2.10%
	CVRP-small		**CVRP-medium**		**CVRP-large**	
	fea.↑	**opt.↓**	**fea.↑**	**opt.↓**	**fea.↑**	**opt.↓**
LLMOPT	100%	2.59%	63%	35.25%	5%	68.98%
DRoC	100%	7.96%	78%	48.45%	9%	96.51%
ORLM	100%	13.83%	97%	163.00%	90%	192.00%
Ours	100%	1.70%	100%	4.57%	100%	7.24%

Our method maintains **100% feasibility across all problem types and scales**, while **feasibility rates for the baselines degrade sharply**—especially on large-scale CVRP (down to 5–9%).

In terms of solution quality, **our optimality gaps remain consistently low**, particularly in medium and large instances, where **model-then-solve pipelines struggle.**

Experiment Results

Benchmark performance

Job Shop Scheduling problem (JSSP)

	TA 15x15		TA 20x15		TA 20x20	
	obj.↓	opt.↓	obj.↓	opt.↓	obj.↓	opt.↓
Best-known solution	1228.9	-	1364.9	-	1617.3	-
SPT	1546.1	25.81%	1813.5	32.87%	2067.2	27.81%
FIFO	1657.4	34.87%	2008.4	47.15%	2297.1	42.03%
ATC	1586.8	29.12%	1794.3	31.46%	2114.1	30.72%
GA* 100x100	1583.8	28.88%	1847.1	35.23%	2304.2	42.47%
L2D greedy	1522.3	23.88%	1813.4	32.86%	2130.2	31.21%
L2D sample	1543.8	25.62%	1779.7	30.39%	2134.2	31.96%
StarJob*	-	19.68%	-	26.91%	-	33.12%
Ours ($N = 8$)	1403.6	14.22%	1653.7	21.16%	1994.2	23.30%
Ours ($N = 64$)	1368.2	11.34%	1600.5	17.26%	1940.9	20.01%

Our language-based JSSP solver outperforms dispatching rules, genetic algorithm, and other learning-based baselines on the Taillard dataset.

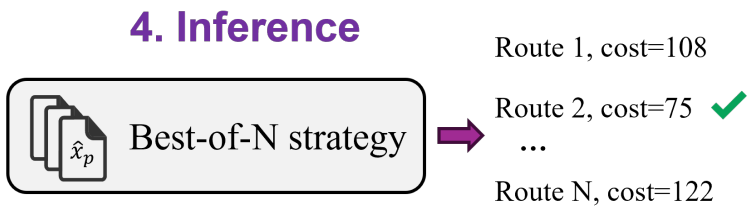
Travelling Salesman problem (TSP)

Instance	AEL	ReEvo	MCTS-AHD	Ours
eil51.tsp	10.84%	6.50%	15.98%	2.02%
berlin52.tsp	12.91%	17.99%	7.08%	4.54%
st70.tsp	4.10%	6.42%	15.45%	0.00%
eil76.tsp	10.26%	7.34%	12.46%	2.91%
pr76.tsp	9.06%	12.17%	10.41%	3.87%
Average Gap	9.43%	10.08%	12.28%	2.69%

Our language-based TSP solver outperforms other LLM-as-programmers method on the representative instances of TSPLIB.

Experiment Results

Scaling Test-time Exploration through BoN



Value	TSP		OP		CVRP		MIS		MVC		PFSP		JSSP	
	fea. ↑	opt. ↓	fea. ↑	opt. ↓	fea. ↑	opt. ↓	fea. ↑	opt. ↓	fea. ↑	opt. ↓	fea. ↑	opt. ↓	fea. ↑	opt. ↓
$N = 1$	82%	3.41%	77%	4.40%	72%	11.80%	52%	1.34%	92%	3.69%	100%	4.48%	99%	23.37%
$N = 2$	97%	2.47%	91%	2.31%	91%	10.51%	57%	1.76%	99%	3.34%	100%	3.61%	100%	20.17%
$N = 4$	100%	1.98%	98%	2.86%	97%	8.84%	68%	2.70%	100%	2.69%	100%	3.03%	100%	18.52%
$N = 8$	100%	1.34%	100%	2.15%	100%	7.24%	75%	2.29%	100%	2.35%	100%	2.62%	100%	16.25%
$N = 16$	100%	1.08%	100%	1.70%	100%	6.04%	76%	2.28%	100%	1.90%	100%	1.98%	100%	14.45%
$N = 32$	100%	0.90%	100%	1.31%	100%	4.76%	85%	2.61%	100%	1.52%	100%	1.85%	100%	13.54%
$N = 64$	100%	0.73%	100%	0.98%	100%	4.30%	92%	2.23%	100%	1.22%	100%	1.54%	100%	11.63%

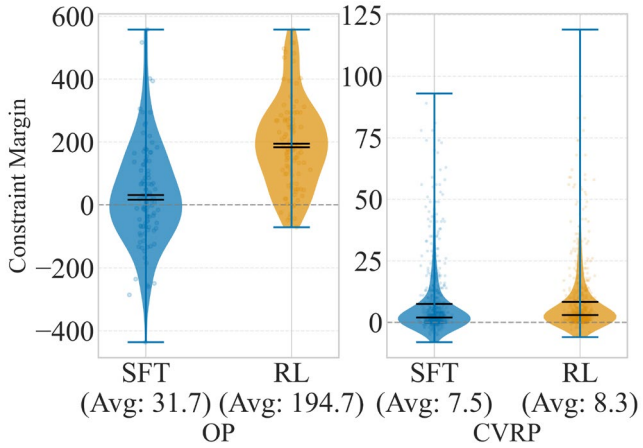
Unlike traditional CO solvers that can **explore multiple branches or backtrack upon reaching dead ends**, autoregressive LLMs commit to each decision and cannot easily revise earlier choices and evaluate constraint satisfaction.

BoN sampling enhances **test-time exploration by generating multiple solution trajectories** and overcome the above issue. Model performance improves consistently with larger values of N.

Experiment Results

The role of RL

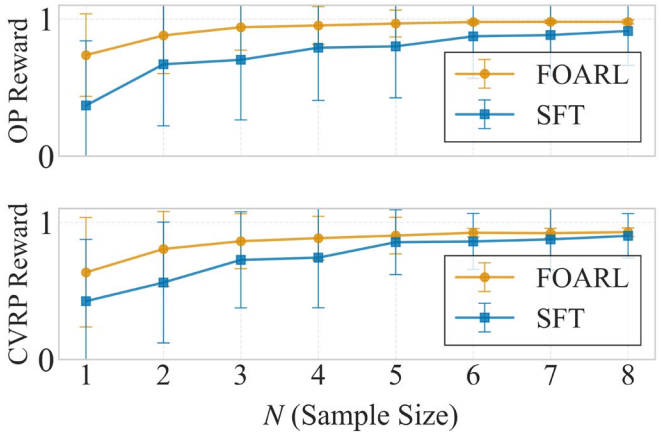
Constraint relaxation operator.



Constraint margin: 1) the difference between actual travel distance and the distance limit in OP, 2) the difference between served demand and vehicle capacity in CVRP.

RL enforces feasibility by **mitigating violations of inequality constraints** (the **constraint margins are improved**), which are common in problems like OP and CVRP.

Sampling efficiency improver.



Models trained with RL require smaller N to reach comparable performance levels as their SFT-only counterparts with larger N.

RL allows the LLM for **achieving better performance with fewer samples**, thereby improving sampling efficiency and reducing test-time computational cost.

Thanks for listening