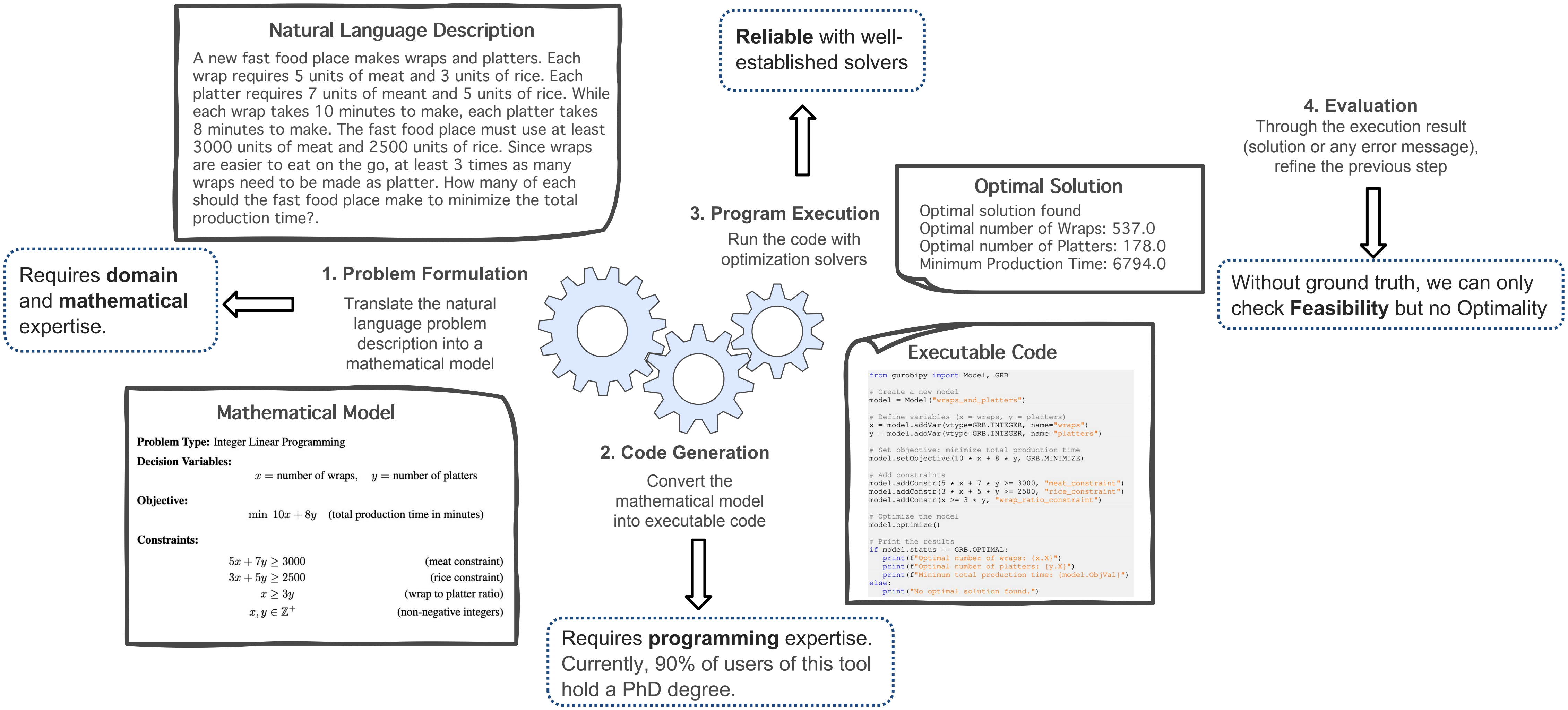
NEURAL INFORMATION PROCESSING SYSTEMS

SolverLLM: Leveraging Test-Time Scaling for Optimization Problem via LLM-Guided Search

Dong Li[1], Xujiang Zhao[2], Linlin Yu[3], Yanchi Li[2], Wei Cheng[2], Zhengzhang Chen[2],
Zhong Chen[4], Feng Chen[5], Chen Zhao[1], Haifeng Chen[2]

[1]Baylor University, [2]NEC Labs America, [3]Augusta University,
[4]Southern Illinois University, [5]University of Texas at Dallas

Solving the Optimization Problem



Natural Language Description

A new fast food place makes wraps and platters. Each wrap requires 5 units of meat and 3 units of rice. Each platter requires 7 units of meant and 5 units of rice. While each wrap takes 10 minutes to make, each platter takes 8 minutes to make. The fast food place must use at least 3000 units of meat and 2500 units of rice. Since wraps are easier to eat on the go, at least 3 times as many wraps need to be made as platter. How many of each should the fast food place make to minimize the total production time?.

1. Problem Formulation

Translate the natural language problem description into a mathematical model

Requires **domain** and **mathematical** expertise.

Mathematical Model

Problem Type: Integer Linear Programming

Decision Variables:

$$x = \text{number of wraps}, \quad y = \text{number of platters}$$

Objective:

$$\min \; 10x + 8y \quad \text{(total production time in minutes)}$$

Constraints:

$$\begin{aligned} 5x + 7y &\geq 3000 && \text{(meat constraint)} \\ 3x + 5y &\geq 2500 && \text{(rice constraint)} \\ x &\geq 3y && \text{(wrap to platter ratio)} \\ x, y &\in \mathbb{Z}^+ && \text{(non-negative integers)} \end{aligned}$$

2. Code Generation

Convert the mathematical model into executable code

Requires **programming** expertise. Currently, 90% of users of this tool hold a PhD degree.

Executable Code

```python
from gurobipy import Model, GRB

# Create a new model
model = Model("wraps_and_platters")

# Define variables (x = wraps, y = platters)
x = model.addVar(vtype=GRB.INTEGER, name="wraps")
y = model.addVar(vtype=GRB.INTEGER, name="platters")

# Set objective: minimize total production time
model.setObjective(10 * x + 8 * y, GRB.MINIMIZE)

# Add constraints
model.addConstr(5 * x + 7 * y >= 3000, "meat_constraint")
model.addConstr(3 * x + 5 * y >= 2500, "rice_constraint")
model.addConstr(x >= 3 * y, "wrap_ratio_constraint")

# Optimize the model
model.optimize()

# Print the results
if model.status == GRB.OPTIMAL:
   print(f"Optimal number of wraps: {x.X}")
   print(f"Optimal number of platters: {y.X}")
   print(f"Minimum total production time: {model.ObjVal}")
else:
   print("No optimal solution found.")
```

3. Program Execution

Run the code with optimization solvers

Reliable with well-established solvers

Optimal Solution

```
Optimal solution found
Optimal number of Wraps: 537.0
Optimal number of Platters: 178.0
Minimum Production Time: 6794.0
```

4. Evaluation

Through the execution result (solution or any error message), refine the previous step

Without ground truth, we can only check **Feasibility** but no Optimality

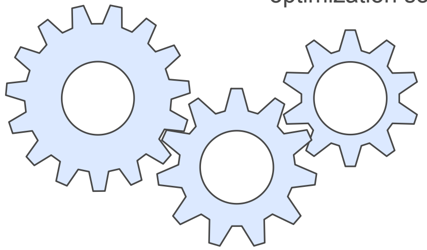
Demands expertise in application domain and mathematical programming

Automating Optimization Problem Solving

Solution: compound AI system

Natural Language Description

A new fast food place makes wraps and platters. Each wrap requires 5 units of meat and 3 units of rice. Each platter requires 7 units of meant and 5 units of rice. While each wrap takes 10 minutes to make, each platter takes 8 minutes to make. The fast food place must use at least 3000 units of meat and 2500 units of rice. Since wraps are easier to eat on the go, at least 3 times as many wraps need to be made as platter. How many of each should the fast food place make to minimize the total production time?.



1. Problem Formulation

Translate the natural language problem description into a mathematical model

Requires **domain** and **mathematical** expertise.

Challenging, but Fruitful

Mathematical Model

Problem Type: Integer Linear Programming

Decision Variables:

$$x = \text{number of wraps}, \quad y = \text{number of platters}$$

Objective:

$$\min \; 10x + 8y \quad \text{(total production time in minutes)}$$

Constraints:

$$\begin{aligned} 5x + 7y &\geq 3000 && \text{(meat constraint)} \\ 3x + 5y &\geq 2500 && \text{(rice constraint)} \\ x &\geq 3y && \text{(wrap to platter ratio)} \\ x, y &\in \mathbb{Z}^+ && \text{(non-negative integers)} \end{aligned}$$

2. Code Generation

Convert the mathematical model into executable code

Requires **programming** expertise. Currently, 90% of users of this tool hold a PhD degree.

Practical and low-risk

Executable Code

```python
from gurobipy import Model, GRB

# Create a new model
model = Model("wraps_and_platters")

# Define variables (x = wraps, y = platters)
x = model.addVar(vtype=GRB.INTEGER, name="wraps")
y = model.addVar(vtype=GRB.INTEGER, name="platters")

# Set objective: minimize total production time
model.setObjective(10 * x + 8 * y, GRB.MINIMIZE)

# Add constraints
model.addConstr(5 * x + 7 * y >= 3000, "meat_constraint")
model.addConstr(3 * x + 5 * y >= 2500, "rice_constraint")
model.addConstr(x >= 3 * y, "wrap_ratio_constraint")

# Optimize the model
model.optimize()

# Print the results
if model.status == GRB.OPTIMAL:
   print(f"Optimal number of wraps: {x.X}")
   print(f"Optimal number of platters: {y.X}")
   print(f"Minimum total production time: {model.ObjVal}")
else:
   print("No optimal solution found.")
```

3. Program Execution

Run the code with optimization solvers

Reliable with well-established solvers

Gurobi

Pyomo

Optimal Solution

```
Optimal solution found
Optimal number of Wraps: 537.0
Optimal number of Platters: 178.0
Minimum Production Time: 6794.0
```

4. Evaluation

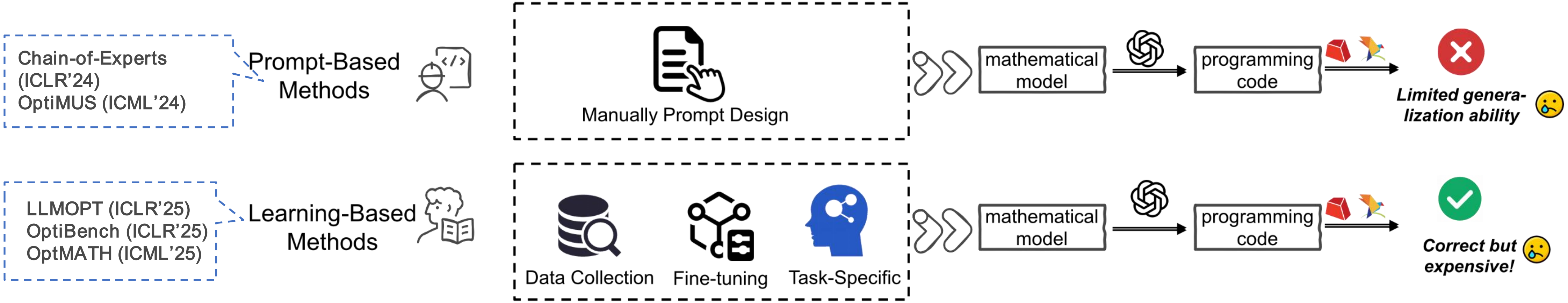
Through the execution result (solution or any error message), refine the previous step

Without ground truth, we can only check **Feasibility** but no Optimality

Not ideal, but the only option

Compound AI: Language Understanding + Reliable Logic Solution

Compound AI System Design (Literature)



Chain-of-Experts (ICLR'24)
OptiMUS (ICML'24)

Prompt-Based Methods

Manually Prompt Design

mathematical model

programming code

Limited generalization ability

LLMOPT (ICLR'25)
OptiBench (ICLR'25)
OptMATH (ICML'25)

Learning-Based Methods

Data Collection

Fine-tuning

Task-Specific

mathematical model

programming code

Correct but expensive!

Prompt-Based Methods

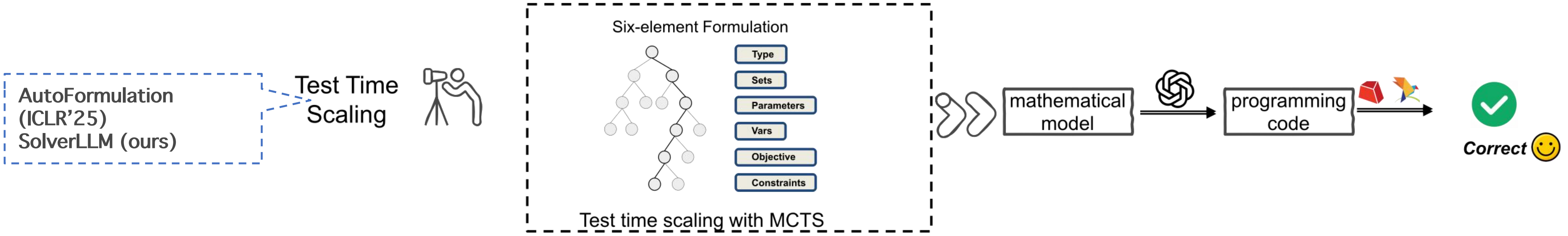
- Coordinate specialized agents under a carefully designed workflow.
- But? Sensitive to prompt choice, making them fragile on unfamiliar optimization tasks.

Learning-Based Methods

- Fine-tune a general LLM on curated problem–solution pairs.
- But? Effectiveness depends on significant dataset labeling and model fine-tuning cost

Automated optimization problem solving is an active research area

Compound AI System Design: Test Time Scaling (TTS)



No need for training ✓

Optimization problems follow established paradigms, and there is abundant unstructured information on the Internet, which is likely part of LLM training data.

✗

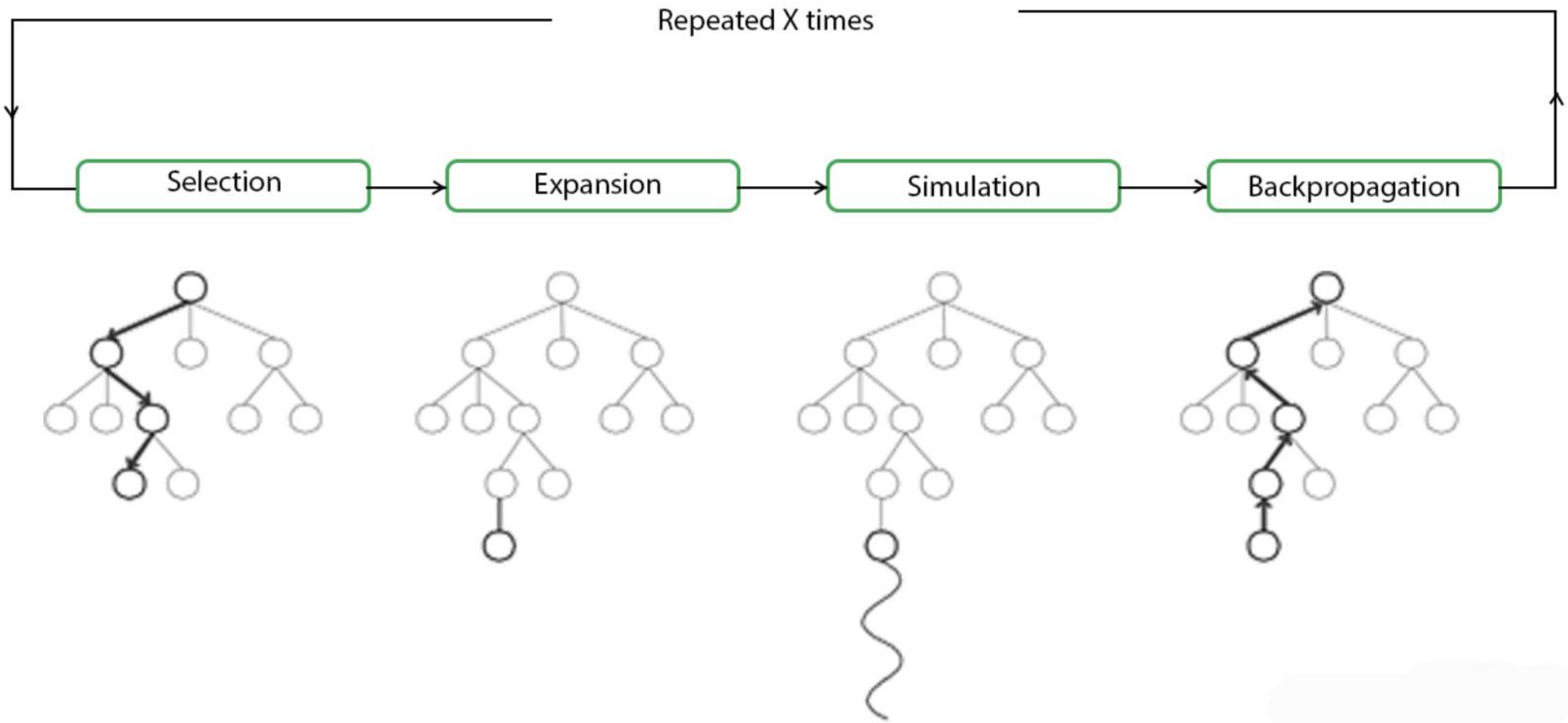
Token consumption and solution exploration are expensive and time-consuming.

Risk of randomness and hallucination during generation.

Model evaluation and backward reasoning are non-trivial without correctness labels.

TTS is promising, but efficiency and reasoning is critical

Monte Carlo Tree Search (MCTS)



Repeated X times

Selection

Expansion

Simulation

Backpropagation

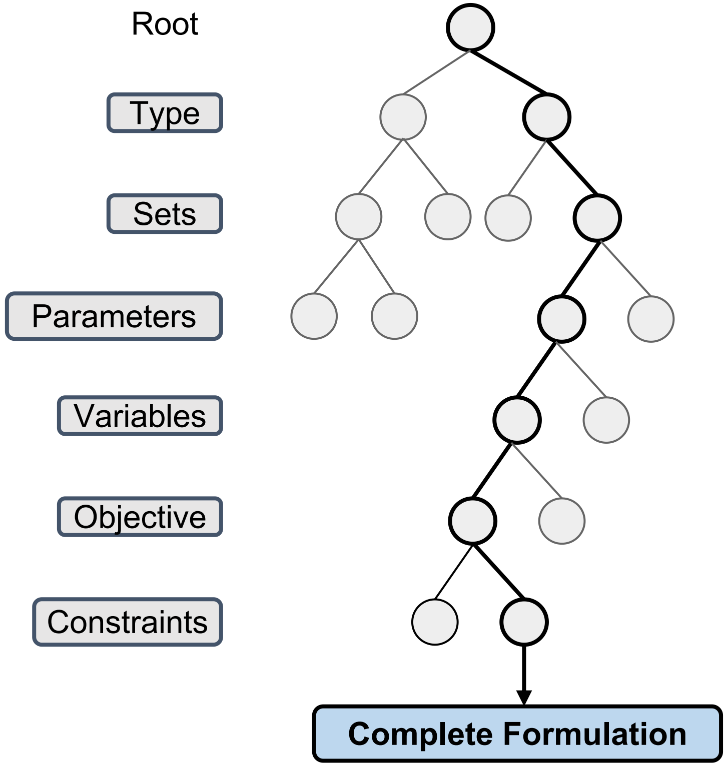
Six-element Schema Formulation for Optimization Problem

Formulation = **Type** + Sets + Parameters + Variables + Objective + Constraints

Global Guidance

Search Tree

- Starting from the root, each level corresponds to an element and is associated with a local expert-guided knowledge base, yielding a tree of depth seven.
- Each node maintains its visit count, accumulated reward, and a trigger that indicates whether it is active.



Root

Type

Sets

Parameters

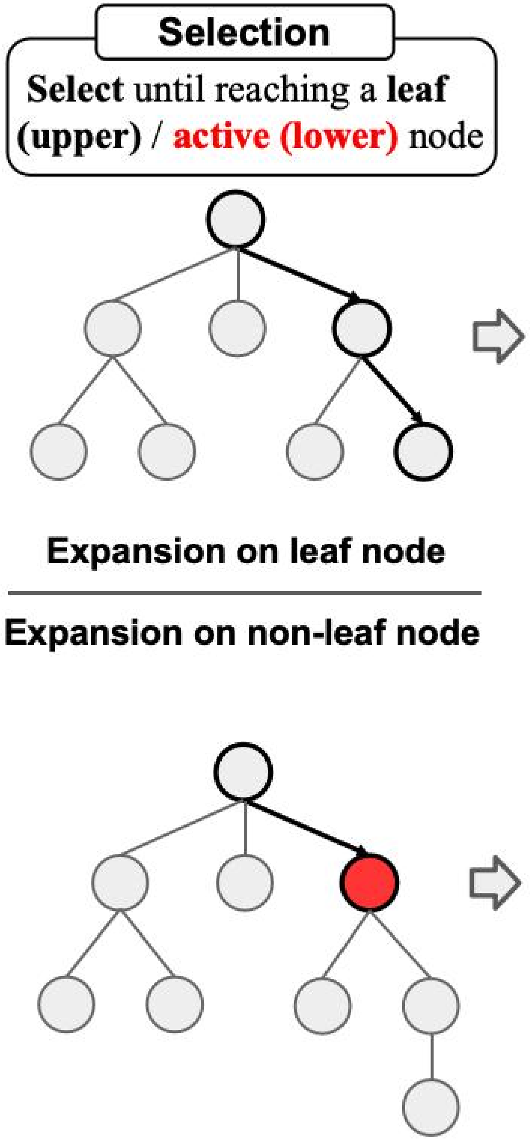
Variables

Objective

Constraints

Complete Formulation

SolverLLM: Optimization Formulation via LLM-Guided MCTS



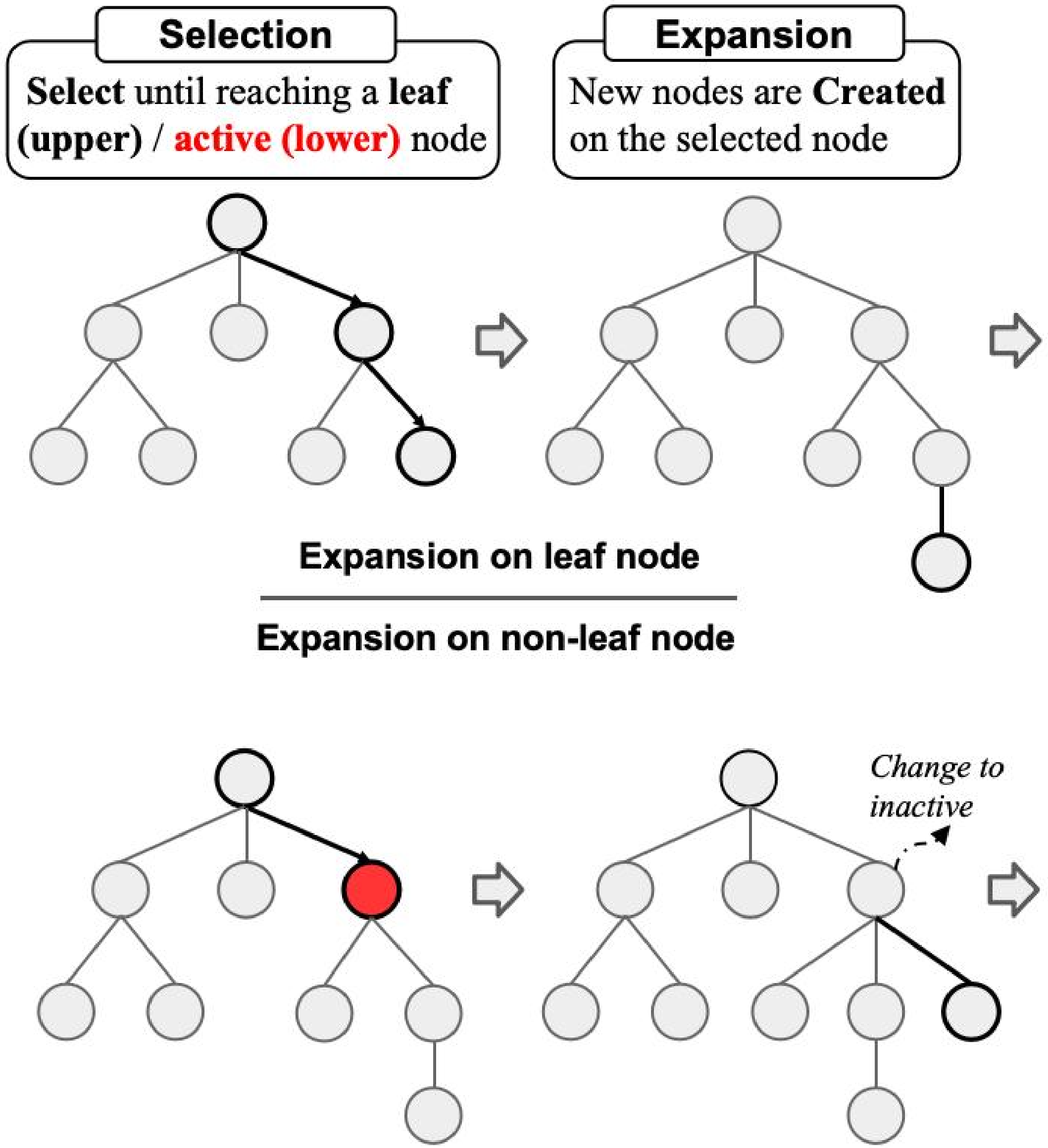
Selection

☐ Starting from the root node, nodes are selected sequentially downward until a leaf or an active non-leaf node is reached.

☐ The selection is guided by the Upper Confidence Bound for Trees (UCT). Given a parent node s, the next child $s_{\text{child}} \in \text{Child}(s)$ is selected according to:

$$s_{child} = \arg \max_{s' \in \text{Child}(s)} \left[\underbrace{Q_{s'}}_{\text{average reward}} + \underbrace{c}_{\text{exploration factor}} \cdot \sqrt{\frac{2 \log \overbrace{N_s}^{\text{visit count}}}{N_{s'}}} \right]$$

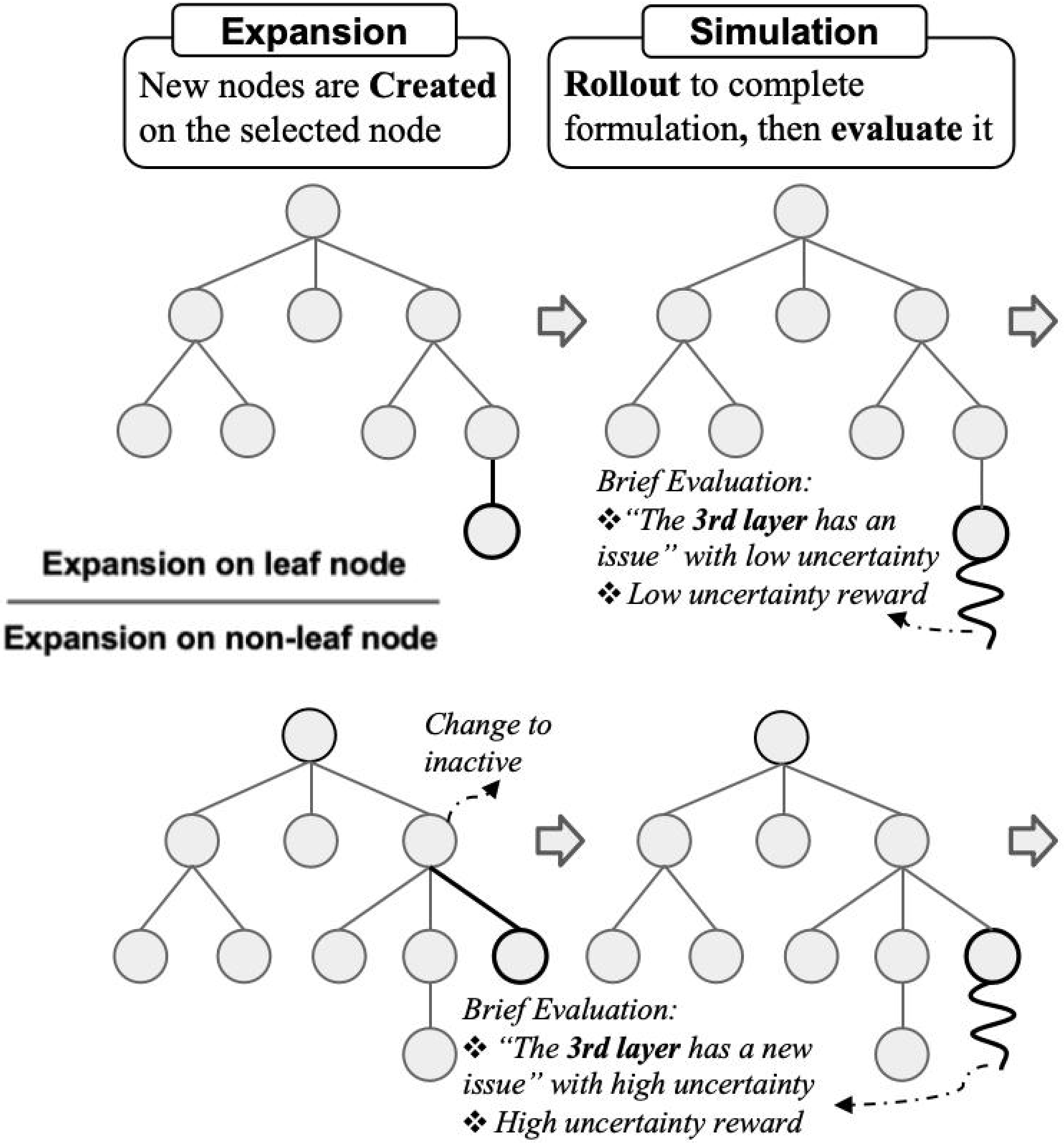
SolverLLM: Optimization Formulation via LLM-Guided MCTS



Dynamic Expansion

- ☐ Expand a new node to the selected leaf or active non-leaf node.
- ☐ LLM-guided expansion with local expert-guided knowledge base.

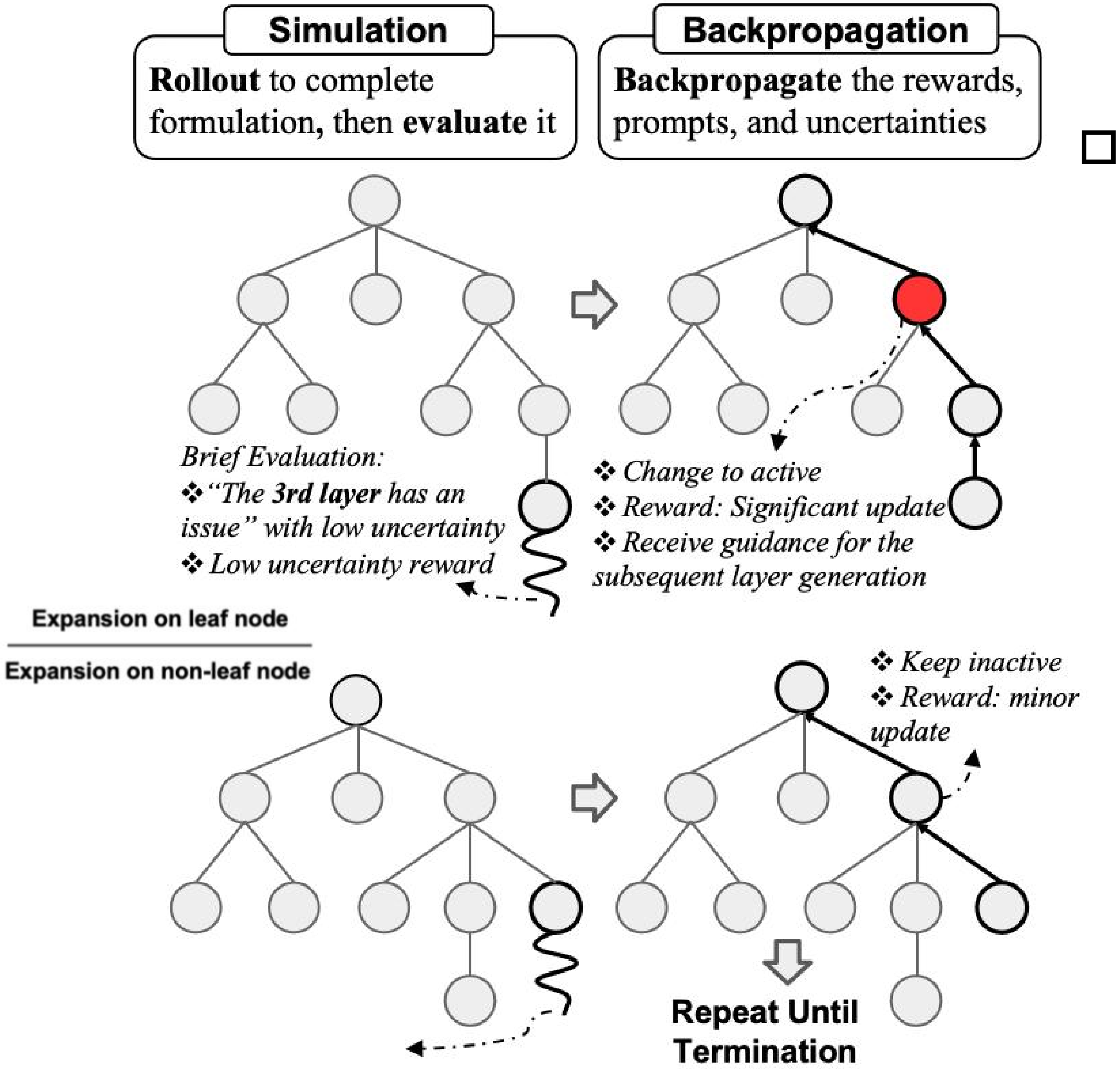
SolverLLM: Optimization Formulation via LLM-Guided MCTS



Simulation

- ☐ Expand continues on expanded node until a complete formulation is obtained, after which alignment and evaluation are performed to yield the following signals:
  - ☐ A reward capturing the factors of feasibility, optimality, and error penalty.
  - ☐ Layer-wise reasoning feedback, which integrates an activation trigger, reasoning guidance, and selection rationale to inform subsequent search and refinement.

SolverLLM: Optimization Formulation via LLM-Guided MCTS

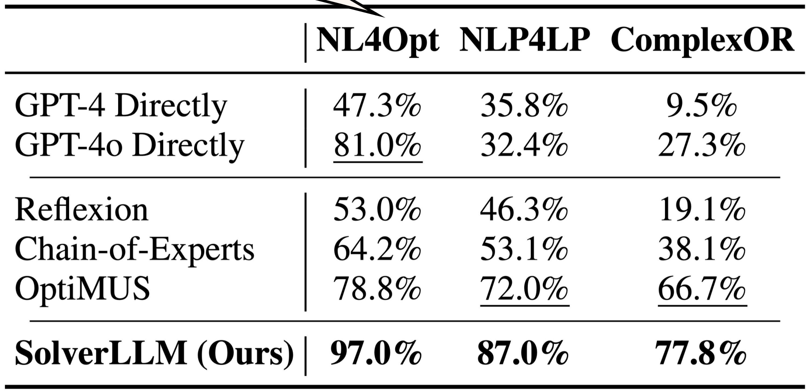


Backpropagation

- ☐ Extend standard MCTS with prompt backpropagation and uncertainty propagation to update the search tree and inform future decisions.
  - ☐ Prompt Backpropagation: The layer-wise reasoning feedback signals are propagated back to each layer, where the local uncertainty is computed via predictive entropy derived from the selection rationale.
  - ☐ Uncertainty Backpropagation: The semantic uncertainty at each evaluated node is estimated as the global uncertainty, which is then used to update each node's reward through an uncertainty-weighted averaging scheme.

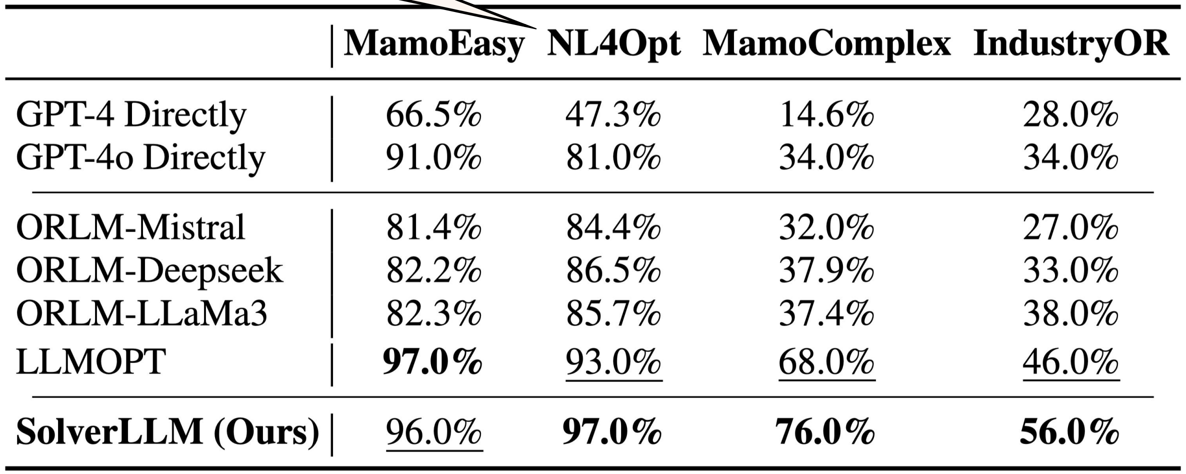
Comparison of Solving Accuracy (SA) Among Different Types of Methods

Prompt-Based methods



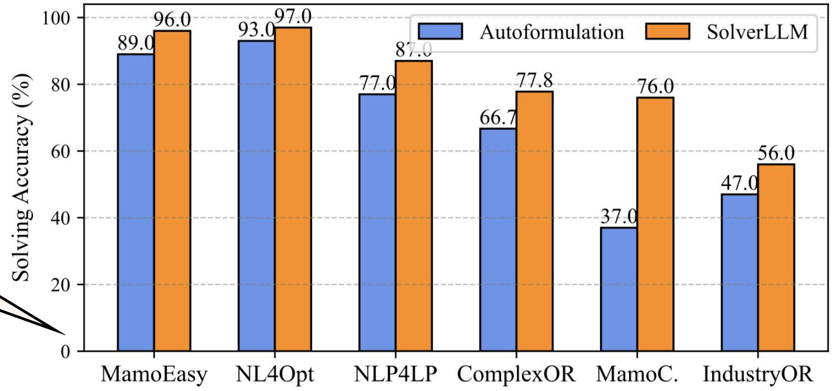
	NL4Opt	NLP4LP	ComplexOR
GPT-4 Directly	47.3%	35.8%	9.5%
GPT-4o Directly	81.0%	32.4%	27.3%
Reflexion	53.0%	46.3%	19.1%
Chain-of-Experts	64.2%	53.1%	38.1%
OptiMUS	78.8%	72.0%	66.7%
SolverLLM (Ours)	**97.0%**	**87.0%**	**77.8%**

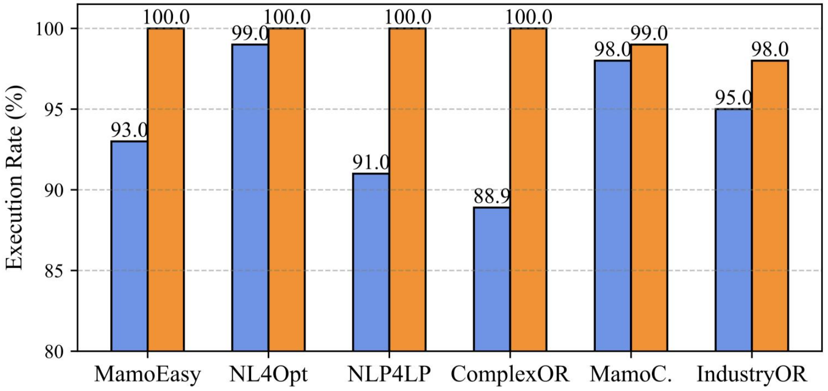
Learning-Based methods



	MamoEasy	NL4Opt	MamoComplex	IndustryOR
GPT-4 Directly	66.5%	47.3%	14.6%	28.0%
GPT-4o Directly	91.0%	81.0%	34.0%	34.0%
ORLM-Mistral	81.4%	84.4%	32.0%	27.0%
ORLM-Deepseek	82.2%	86.5%	37.9%	33.0%
ORLM-LLaMa3	82.3%	85.7%	37.4%	38.0%
LLMOPT	**97.0%**	93.0%	68.0%	46.0%
SolverLLM (Ours)	96.0%	**97.0%**	**76.0%**	**56.0%**

Test-Time Scaling method





Thanks for Your Wathcing!