HeurAgenix: Leveraging LLMs for Solving Complex Combinatorial Optimization Challenges

arxiv'25

Xianliang Yang[1] **Ling Zhang**[1] **Haolong Qian**[1,2] **Lei Song**[1] **Jiang Bian**[1]
[1]Microsoft Research Asia, Beijing, China
[2]Tsinghua University, Beijing, China
{Xianliang.Yang, Ling.Zhang, v-haolqian, Lei.Song, Jiang.Bian}
@microsoft.com

https://github.com/microsoft/HeurAgenix

Presenter：Changshuo Yao

2025.12.29

作者团队

- 微软亚洲研究院（Microsoft Research Asia）助理常务董事（Assistant Managing Director）
- 北京大学学士学位
  佐治亚理工学院计算机科学博士学位
- 研究方向： AI for Industry
- https://www.microsoft.com/en-us/research/people/jiabia/

Jiang Bian

背景知识：组合优化（Combinatorial Optimization）

- 组合优化：在**离散定义域**上对目标函数**求最值**的最优化问题
- 例如：
  - TSP（旅行商问题，Travelling Salesman Problem）
  - CVRP（带容量约束的车辆路径问题，Capacitated Vehicle Routing Problem）
  - MKP（多维背包问题，Multidimensional Knapsack Problem）
  - JSSP（作业车间调度问题，Job Shop Scheduling Problem）

背景知识：旅行商问题（Traveling Salesman Problem）

- 直观表述：一位旅行商需要访问一系列城市，每个城市恰好访问一次，最终返回出发的城市，并要求总行程路径最短
- 图论表述：在一个完全加权图 $G = (V, E)$ 中寻找一条权重最小的哈密顿回路（经过每一个顶点恰好一次，并最终回到起点的闭合路径）
- 是NP-Hard的

背景知识：组合优化（Combinatorial Optimization）

- 背景知识
  - 组合优化
  - TSP问题
  - 组合优化的难点
  - 超启发式算法
- 设计框架
- 实验
- Thinking

- 组合优化问题的难点
  - 搜索空间呈指数级增长，传统精确算法计算上难以处理大规模实例
  - 故常使用**启发式算法**

- 启发式算法的限制：
- ① 高度依赖具备专业知识的专家进行手工设计
- ② 单一启发式的泛化能力有限，无法跨不同实例有效迁移
- ③ 难以适应动态变化的问题条件

背景知识：超启发式（Hyper-heuristic）算法

- 超启发式（Hyper-heuristic）算法
- 核心思想：
  - 不再直接求解问题本身（例如“寻找最短路径”）
  - 而是搜索“求解该问题的最佳算法”（例如“寻找最好的寻路规则”）。
- 两种方法
  - 生成型超启发式（Generation Hyper-heuristics）
    - 通过组合基本组件（常借助遗传算法或遗传编程）自动生成新的启发式规则。
  - 选择型超启发式（Selection Hyper-heuristics）
    - 根据当前问题状态，从预定义的启发式集合中动态选择最合适的一个。
- 机遇
  - 利用LLM同时自动化实现启发式的生成与选择

背景知识：超启发式（Hyper-heuristic）算法

- 当前主流超启发式方法（基于LLM）
  - 代表性工作：FunSearch、EoH、ReEvo
  - 工作流程：利用LLM编写单个启发式规则的代码，再将其嵌入标准外部求解器（如引导式局部搜索GLS或蚁群算法ACO）中使用。
  - 一旦生成启发式规则，其在整个求解过程中固定不变，无法随问题状态的变化而动态调整。

方法	演化方法	求解策略	依赖外部求解器
AlphaEvolve	进化算法	静态	×
FunSearch / EoH / ReEvo	进化+LLM	静态	√
HeurAgenix（本文）	进化+LLM	动态	×

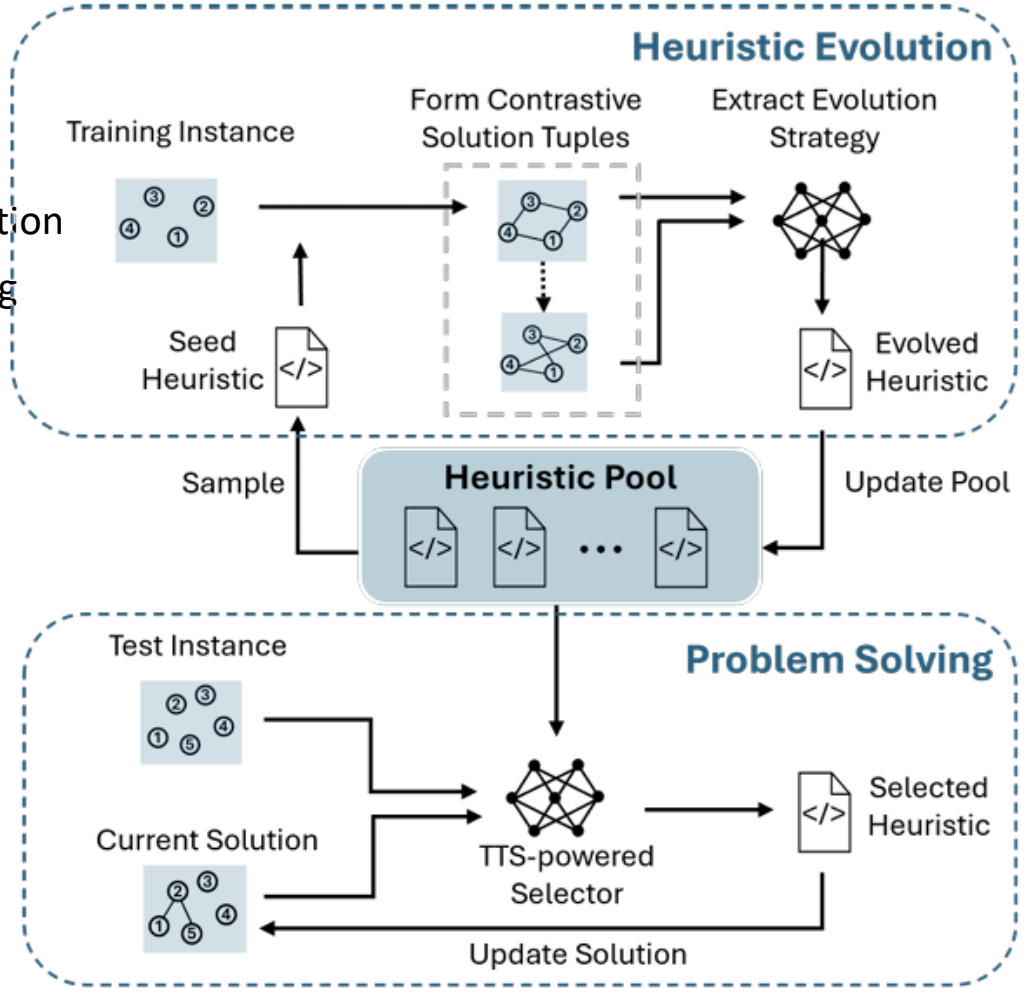
设计框架

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- **HeurAgenix的研究动机：**
- 两个主要问题：
  - 1. 如何自动设计出好的启发式算法？
  - 2. 在真实的求解过程中，如何在不同启发式算法之间切换？
- 一个次要问题：
  - 在问题求解阶段，能否通过微调小LLM来优化推理成本？

设计框架

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

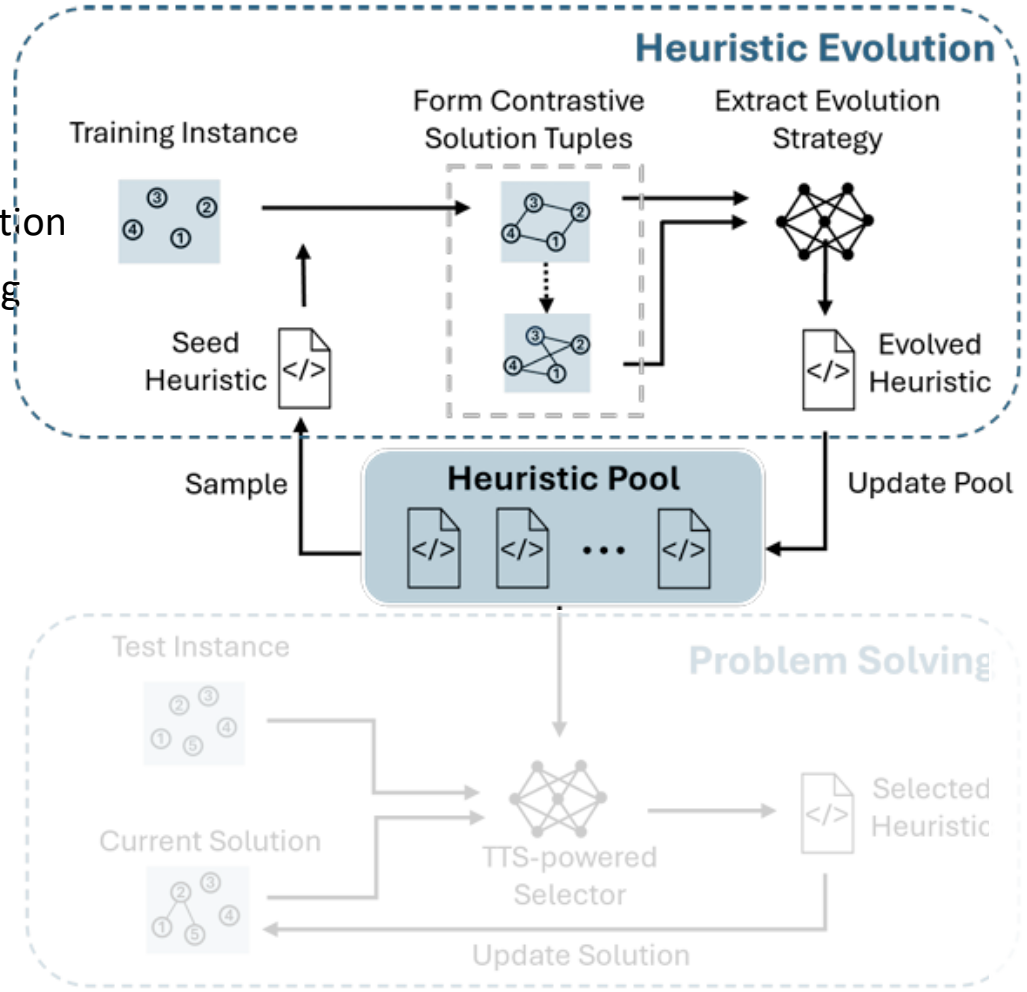


- ①Heuristic Evolution:
- 离线
- 针对某类问题（比如TSP），自动设计和改进启发式算法，存储于Pool中备用

- ②Problem Solving:
- 在线
- 求解时，根据当前的问题状态，从Pool中动态选择最合适的启发式算法

设计框架

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking



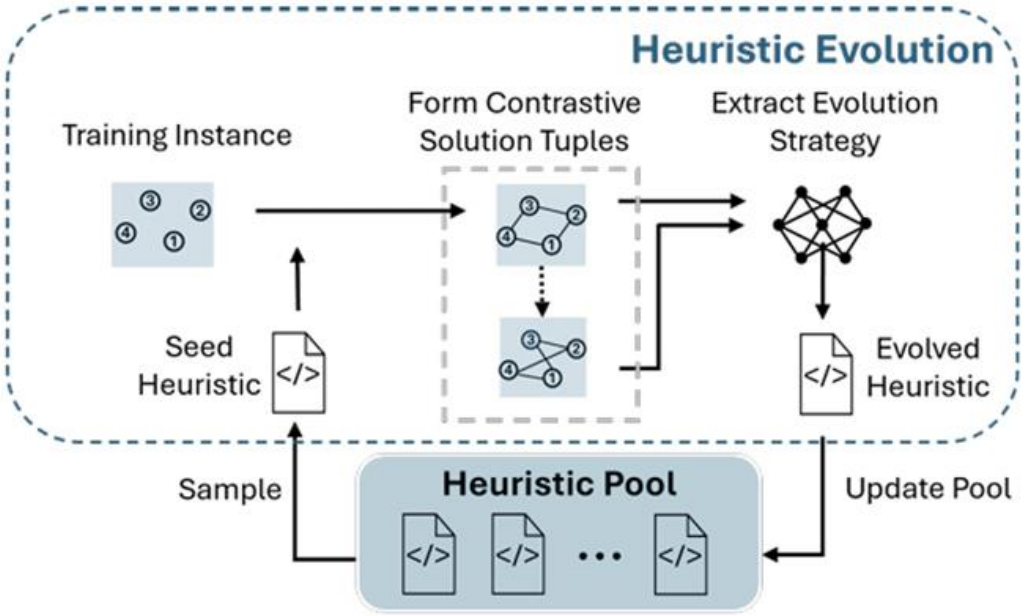
- ①Heuristic Evolution:
- 离线
- 针对某类问题（比如TSP），自动设计和改进启发式算法，存储于Pool中备用

- ②Problem Solving:
- 在线
- 求解时，根据当前的问题状态，从Pool中动态选择最合适的启发式算法

设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 输入：问题实例 d “种子”启发式算法（seed heuristic）
- Step1 生成基本解
- Step2 生成对比解
- Step3 识别关键操作
- Step4 提取进化策略
- Step5 迭代精炼

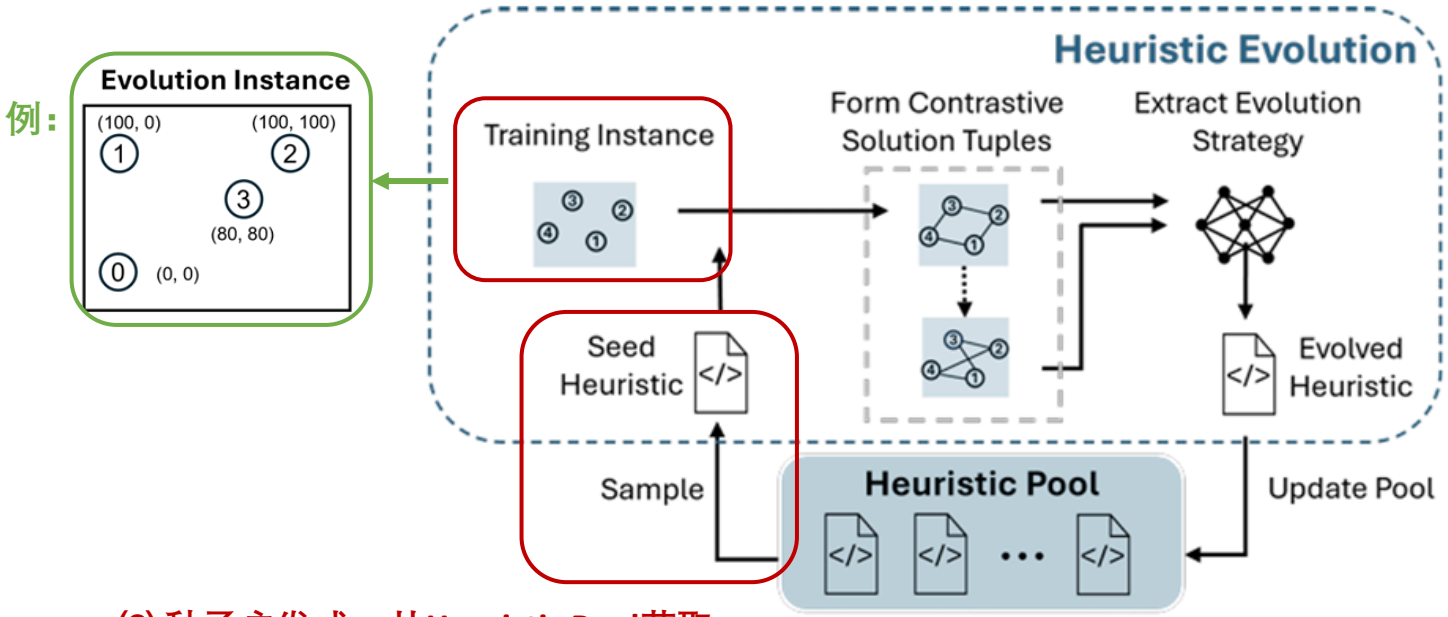


设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 输入：(1) 问题实例 (2) 种子启发式

(1) 问题实例：从训练实例集中获取



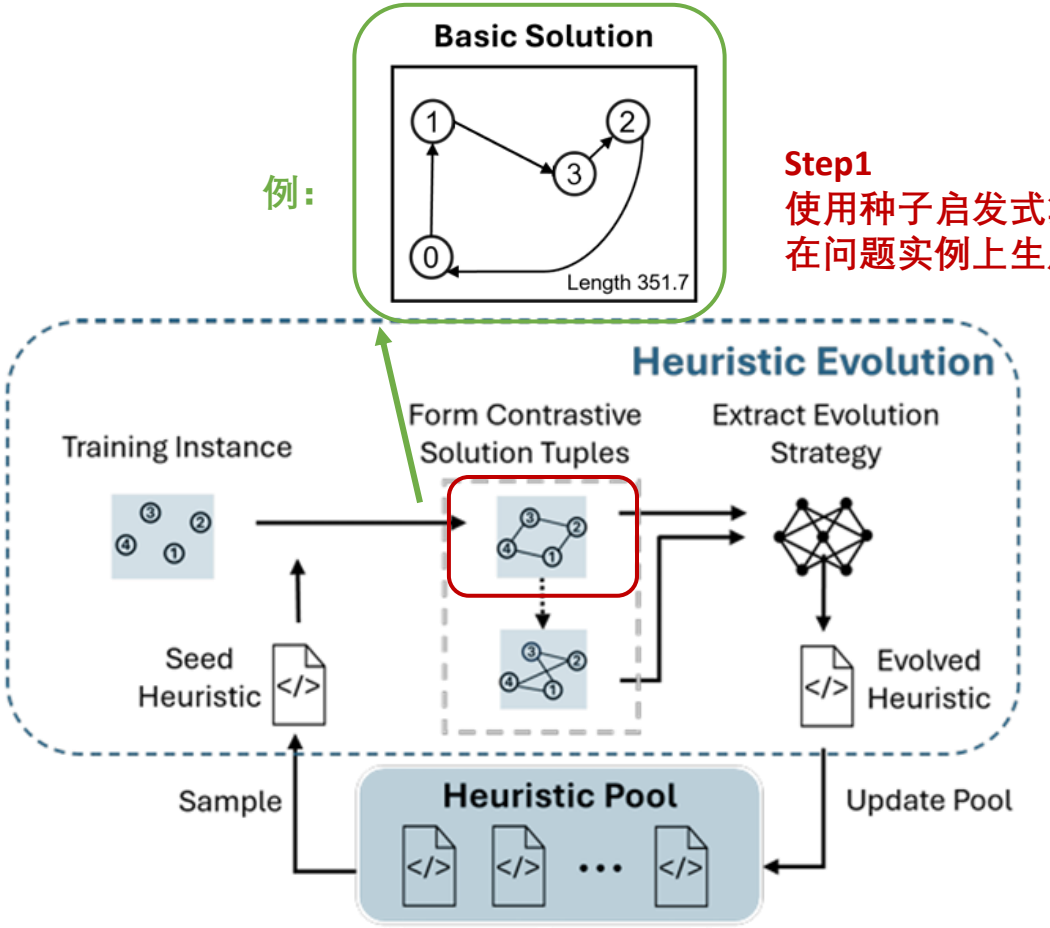
(2) 种子启发式：从Heuristic Pool获取
若Pool为空，则可以构造简单方法（如贪心）

例：Nearest Neighbor (每次选最近的)
以Python函数形式表述

设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- Step1 生成基本解



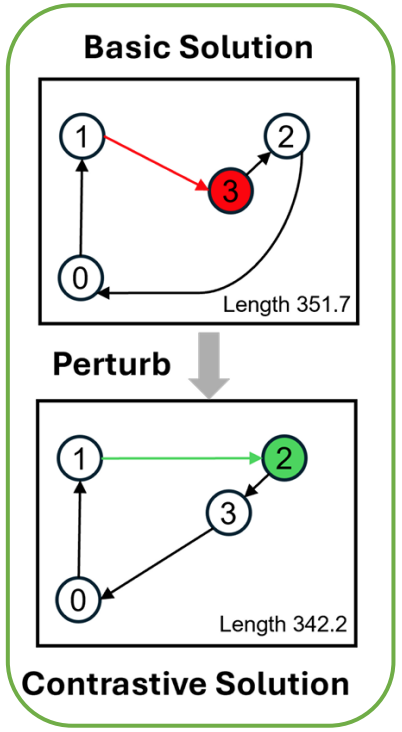
Step1
使用种子启发式算法
在问题实例上生成基础解决方案

设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

Step2 生成对比解

例：

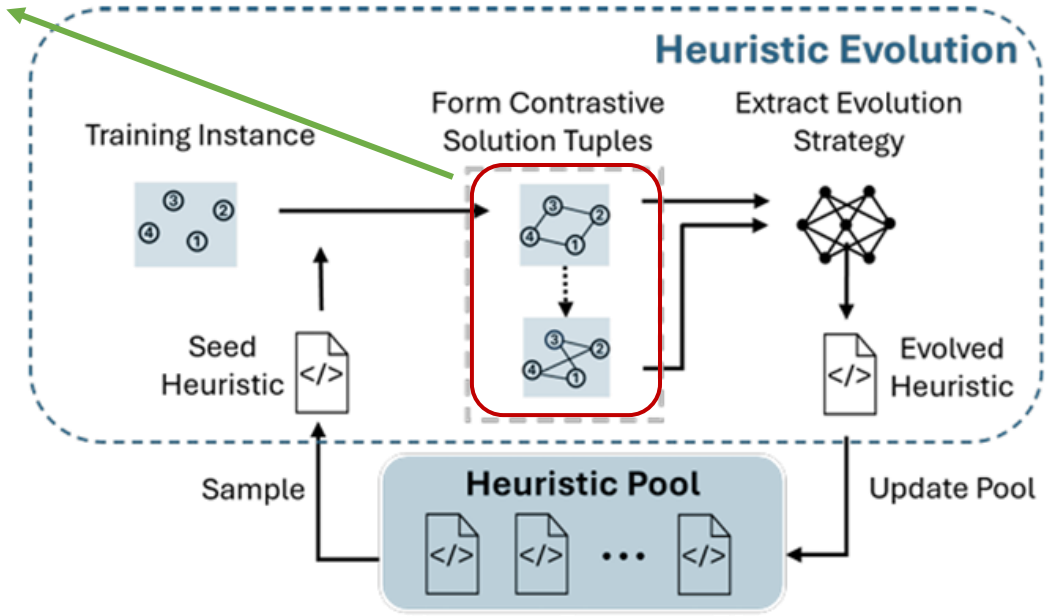


Step2

通过扰动基本解来生成潜在的更优对比解

扰动：随机选择基本解中的一部分操作，将这些操作替换为其他可行的替代操作

进行最多P次扰动实验，若新解更优，则保存为对比解

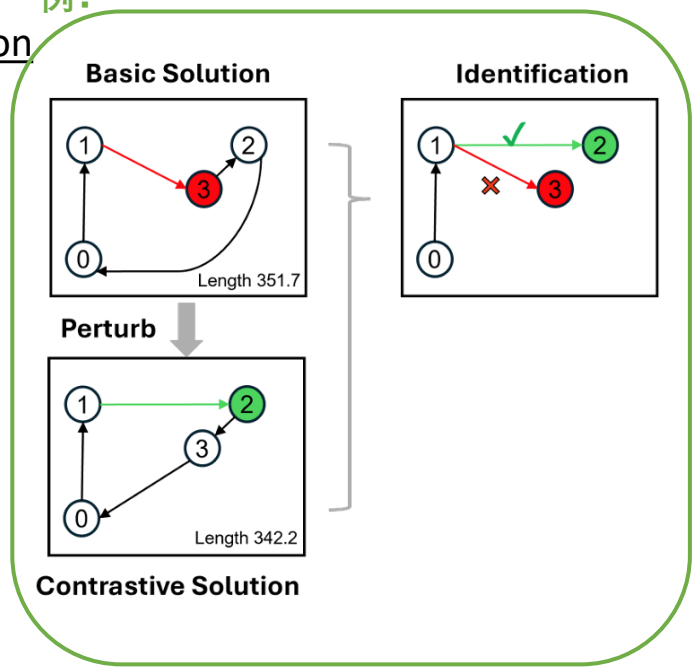


设计框架 - ① Heuristic Evolution

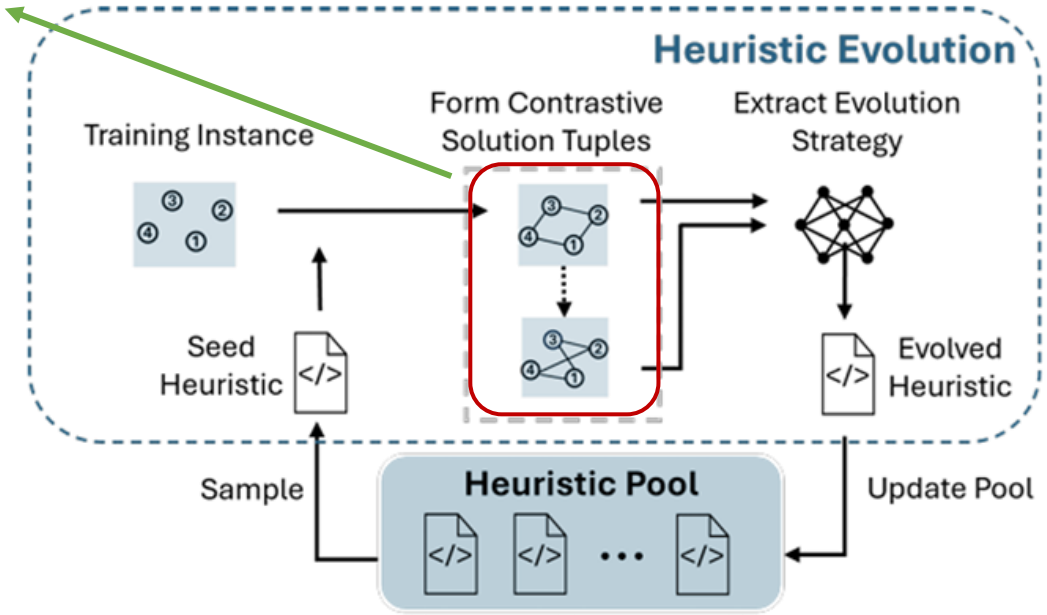
- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- Step3 识别关键操作

例：



Step3
并非所有Step2中修改的操作都对性能提升有同等贡献
故通过独立测试每个修改的操作，识别出最关键的一个操作

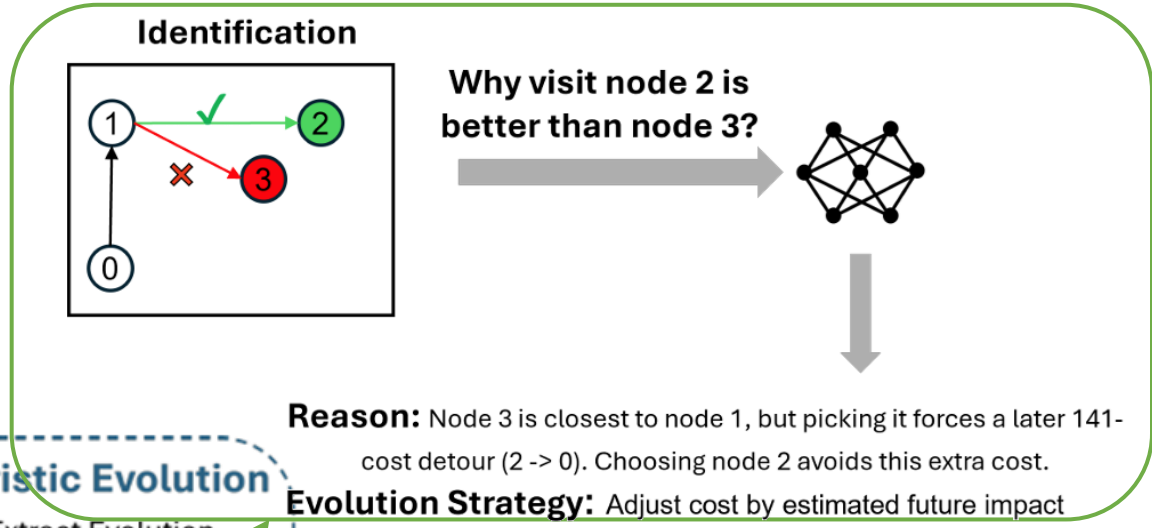


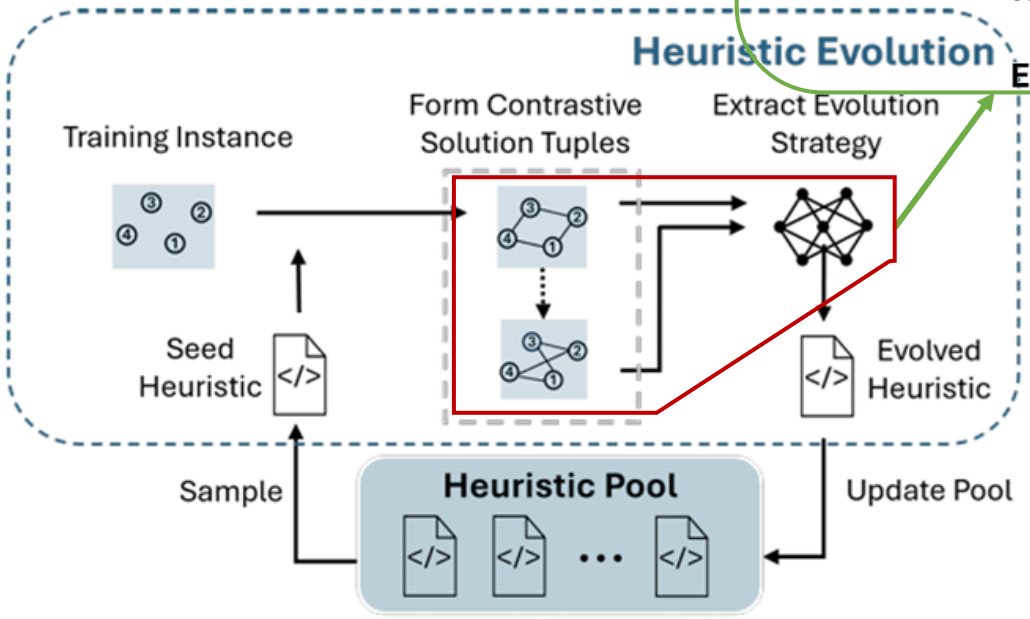
设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- Step4 提取进化策略

例：





Step4
使用LLM分析：为什么Step3中修改的操作能带来性能提升
从具体案例中抽象出可泛化的改进原则，
输出一个高层次的进化策略E

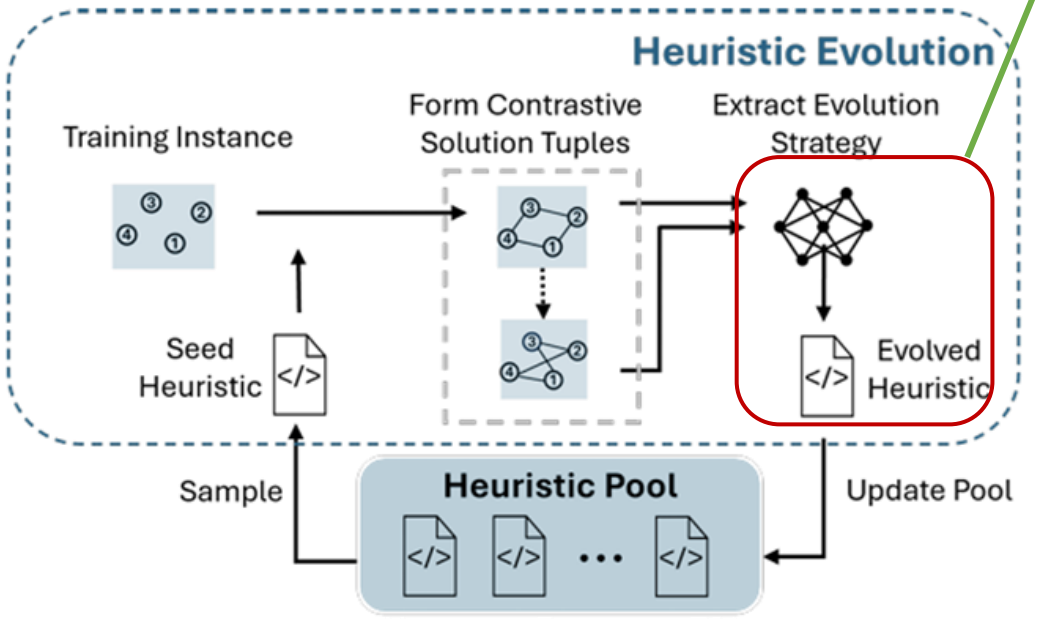
设计框架 - ① Heuristic Evolution

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- Step5 迭代精炼

例：

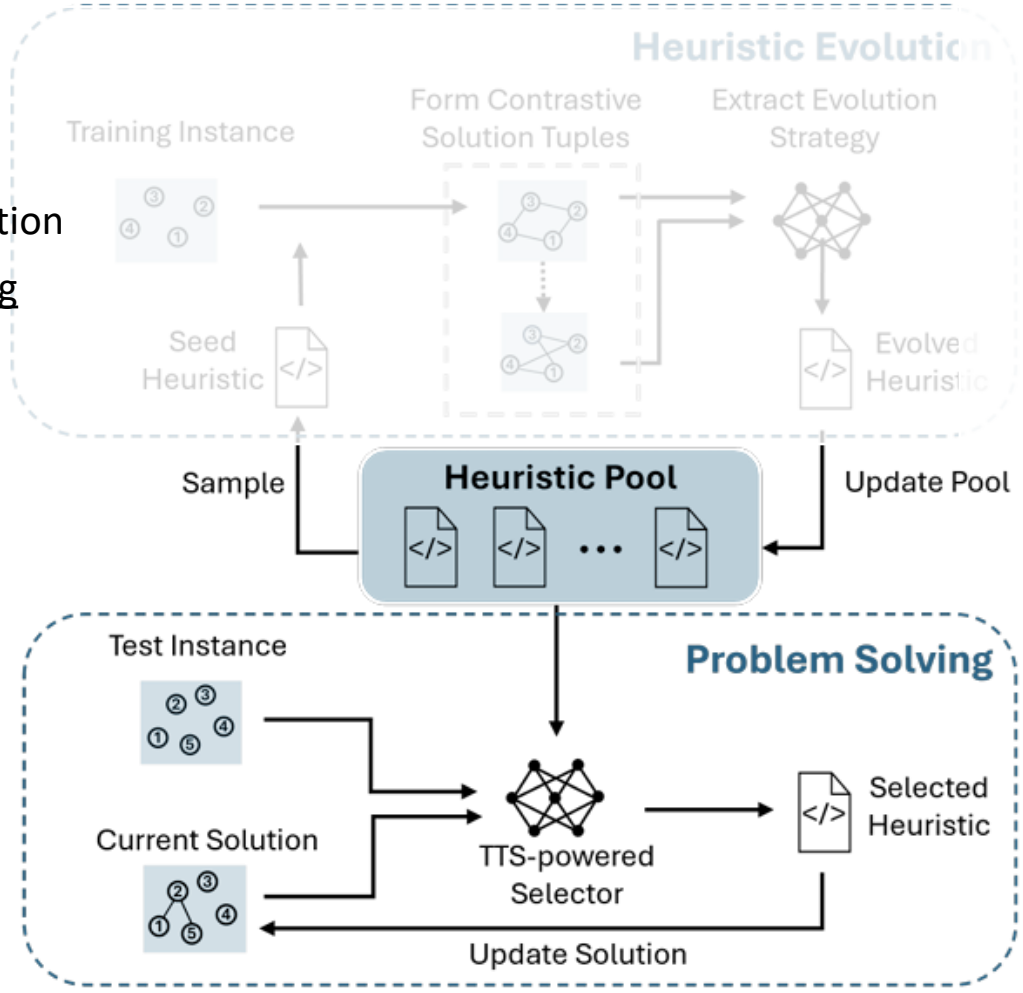
- H_0 ← 初始启发式，I ← 0
- While 迭代次数 i < i_max && 迭代产生改进：
  - P_i ← 测量 H_i 性能
  - H_{i+1} ← LLM改进(H_i)
  - P_{i+1} ← 测量 H_{i+1} 性能
  - 比较 P_i P_{i+1} 判断是否产生改进



Step5
将提取的进化策略E应用到原始启发式上
并通过反复评估和改进来精炼算法（Python代码）
直到性能不再提升或者到达最大次数

设计框架

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

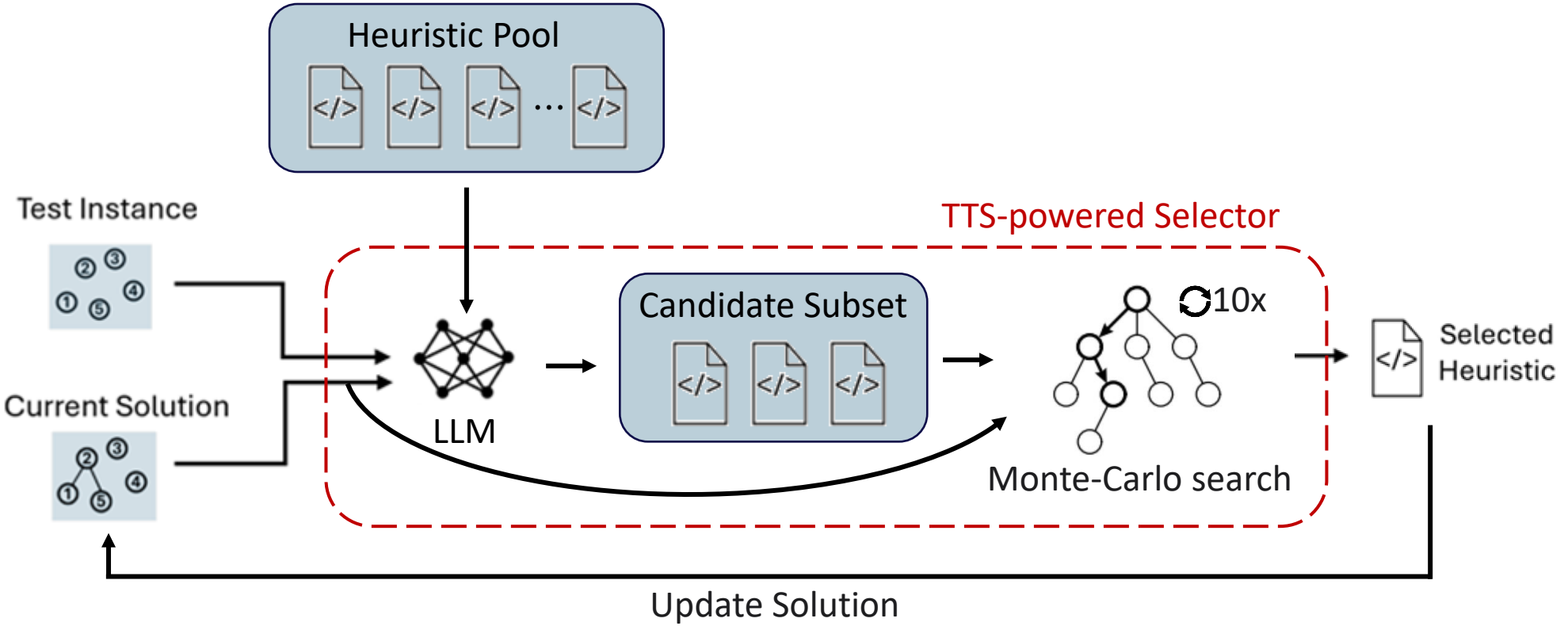


- ①Heuristic Evolution:
- 离线
- 针对某类问题（比如TSP），自动设计和改进启发式算法，存储于Pool中备用

- ②Problem Solving:
- 在线
- 求解时，根据当前的问题状态，从Pool中动态选择最合适的启发式算法

设计框架 - ② Problem Solving

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking



- Step1 LLM初步筛选
- Step2 蒙特卡洛搜索
- Step3 选取最佳启发式 连续执行M次

设计框架 - ② Problem Solving

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

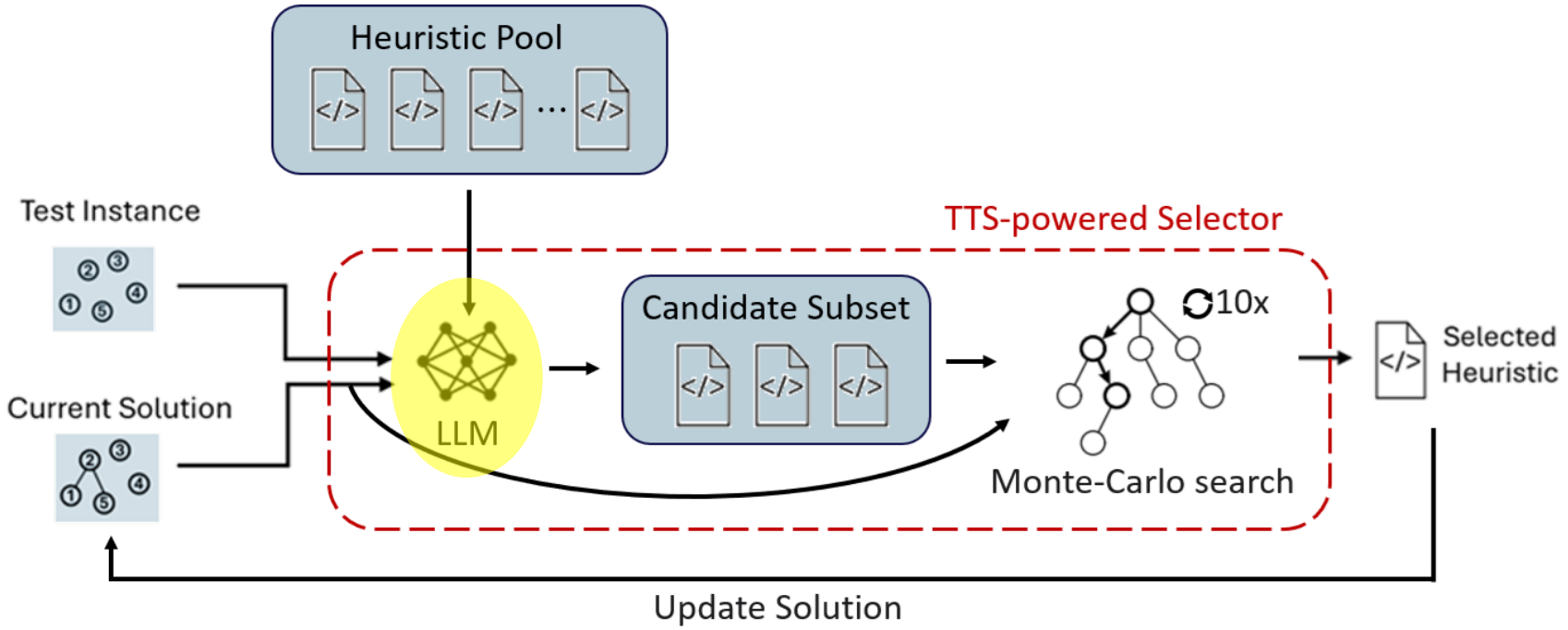
- TTS: Test time Scaling
- TTS 指在推理阶段（测试时）主动增加计算资源以提升输出质量的技术。
- 例如，在LLM推理中的TTS举例
  - 并行（Parallel Scaling）：生成多个候选答案（Best-of-N），然后选择最佳一个。
  - 顺序（Sequential Scaling）：使用Chain-of-Thought（CoT），花费更多Token让模型具备逐步思考、自我修正的能力。

- 本文中TTS应用于蒙特卡洛部分，而不是LLM
  - 通过增加模拟次数来提高决策质量。

设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 可通过微调轻量级模型 (Qwen-7B) 作为选择器
- 降低推理成本，提高求解的速度和实时性



设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 微调方法：GRPO （Group Relative Policy Optimization）
  - 一种强化学习方法，不需要标注数据，只需要能“验证”好坏(reward)。
  - 应用场景：没有标注数据但能够验证输出时。

- 本文针对Qwen-7B微调的设计
  - (1) 离线数据收集
  - (2) 双重奖励机制

设计框架 - ② Problem Solving – 模型微调

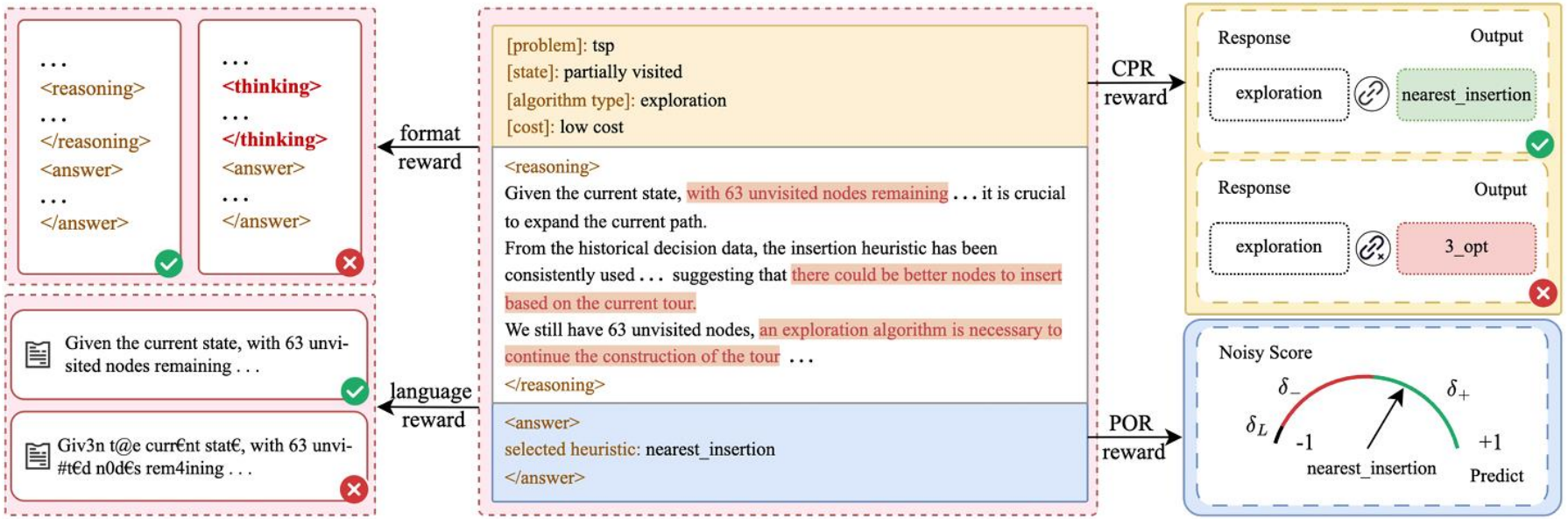
- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- (1) 离线数据收集
- 数据格式： (z, H, QH)
  - z：当前问题状态
  - H：从启发式池中抽取的候选启发式
    - （一组启发式）
  - QH：通过蒙特卡洛搜索评估的启发式价值
    - （这组启发式中最好的启发式的价值）
- 数据收集方式
  - 在使用GPT4o求解问题的过程中收集
  - 包括两种轨迹：
    - 贪心轨迹：每一步选择最高QH值的启发式
    - 随机轨迹：每一步选择随机启发式，使得选择器能够从非最优情境中恢复

设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

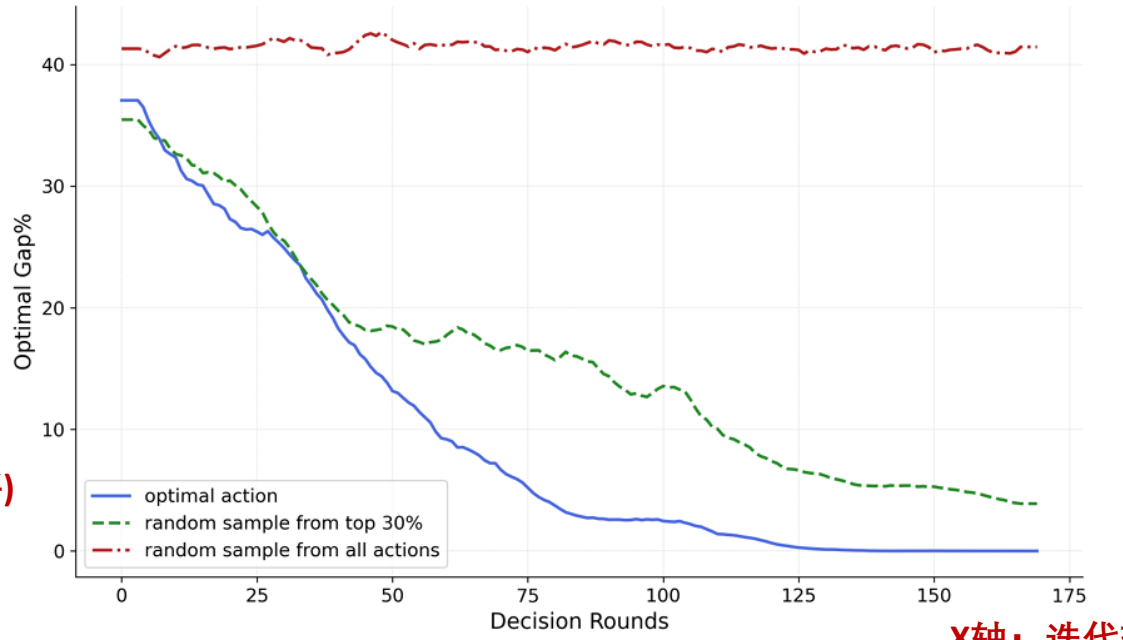
- 挑战：组合优化问题中评估信号噪声大
- (2) 双重奖励机制
  - 基于偏好的奖励 Preference-based Outcome Reward (POR)
  - 上下文感知奖励 Context-PerceptionReward (CPR)



设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 基于偏好的奖励 Preference-based Outcome Reward (POR)
- 观察：
  - 从top 30%启发式中均匀随机选择，几乎与总是选择最佳启发式一样好，且显著优于随机选择。



Y轴：与最优解的gap (越小越好)

X轴：迭代轮次

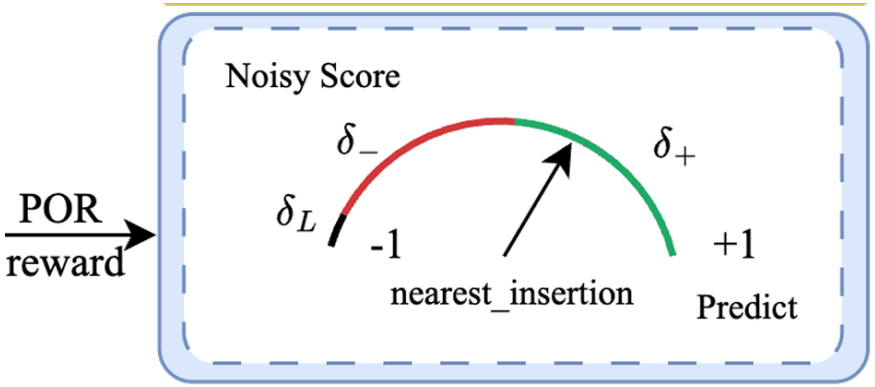
设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 基于偏好的奖励 Preference-based Outcome Reward (POR)
- 设计：
  - 压缩内部差异：减少内部排名的细微变化带来的影响
  - 放大边界差异：放大边界处的不连续性
  - 错误区域惩罚：为跨越到错误区域的启发式提供固定惩罚

$$\ell = \mathrm{rank}\big(z, \hat{H}, \{Q_H\}_{H\in\mathcal{H}}\big),$$

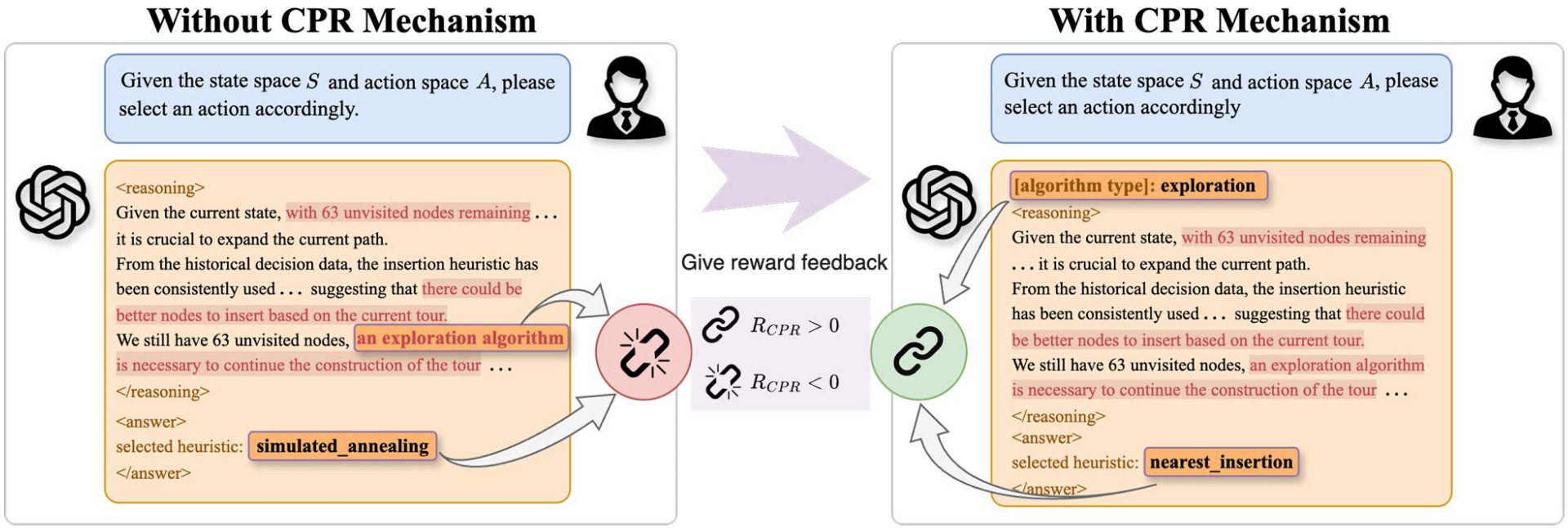
$$R_{\mathrm{POR}}(z,\hat{H}) = \begin{cases} R_p\left(1 - \dfrac{\ell-1}{n_{\mathrm{pos}}}\right), & 1 \le \ell \le n_{\mathrm{pos}}, \\ -R_n\left(\dfrac{\ell - n_{\mathrm{pos}}}{n_{\mathrm{neg}} - n_{\mathrm{pos}}}\right), & n_{\mathrm{pos}} < \ell \le n_{\mathrm{neg}}, \\ -R_L, & n_{\mathrm{neg}} < \ell \le n. \end{cases}$$



设计框架 - ② Problem Solving – 模型微调

- 背景知识
- 设计框架
  - 整体设计
  - ①Heuristic Evolution
  - ②Problem Solving
    - 微调
    - 数据收集
    - 奖励设计
- 实验
- Thinking

- 上下文感知奖励 Context-PerceptionReward (CPR)
- 设计：
  - 奖励模型对环境的理解能力，而非仅仅关注最终的优化结果。



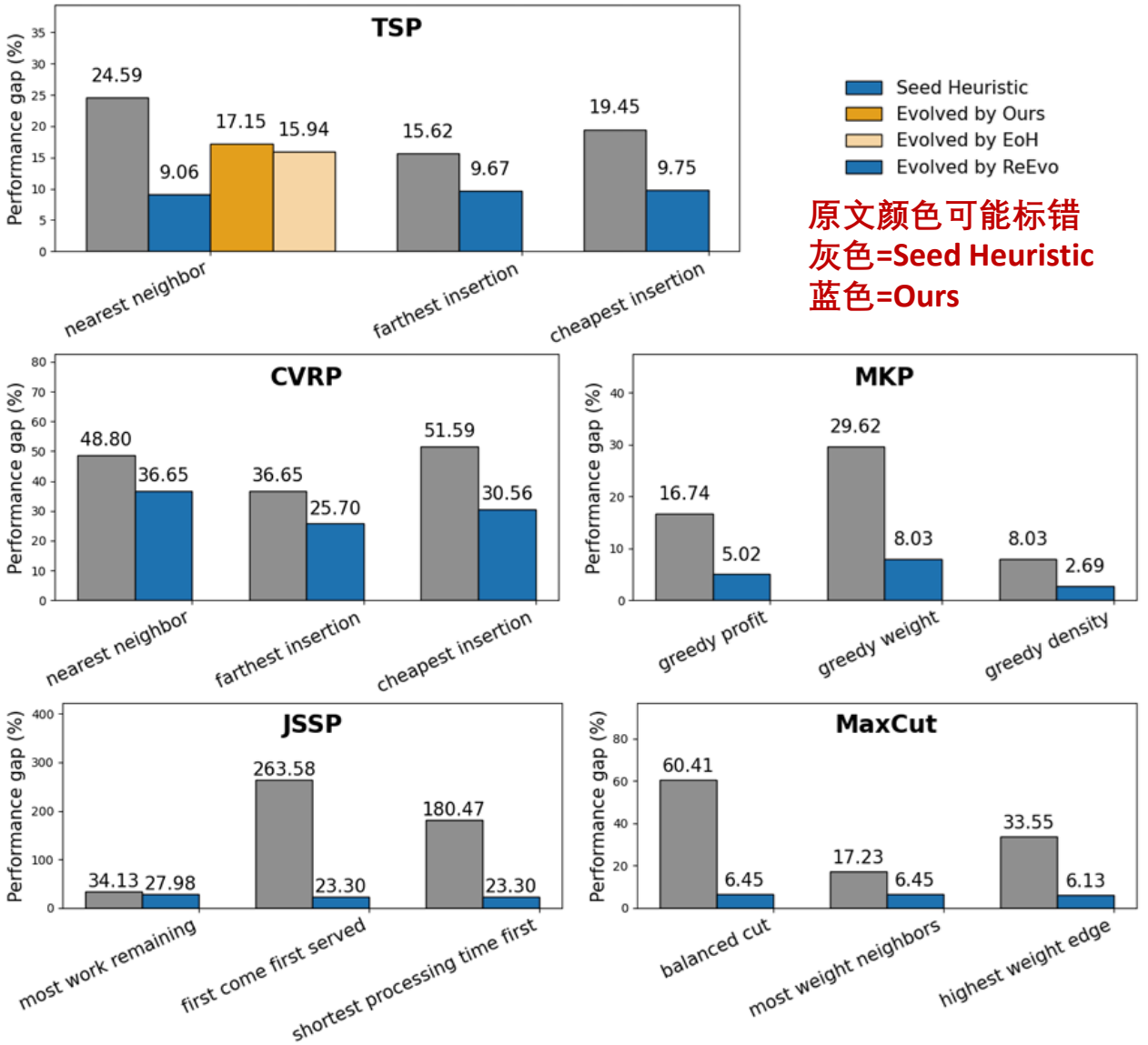
实验

- 硬件：
  - Intel Xeon CPU，NVIDIA RTX A6000(48G) GPU
- LLM：
  - 演化阶段：GPT-4o
  - 选择阶段：GPT-4o或微调Qwen-7B
- 数据集：
  - 5种CO问题(TSP、CVRP、MKP、JSSP、MaxCut)
- 评价指标：
  - 与最优解的性能差距（越小越好）

实验

- 背景知识
- 设计框架
- 实验
  - 进化实验
  - 求解实验
  - LLM微调性能实验
  - 蒙特卡洛搜索TTS实验
- Thinking

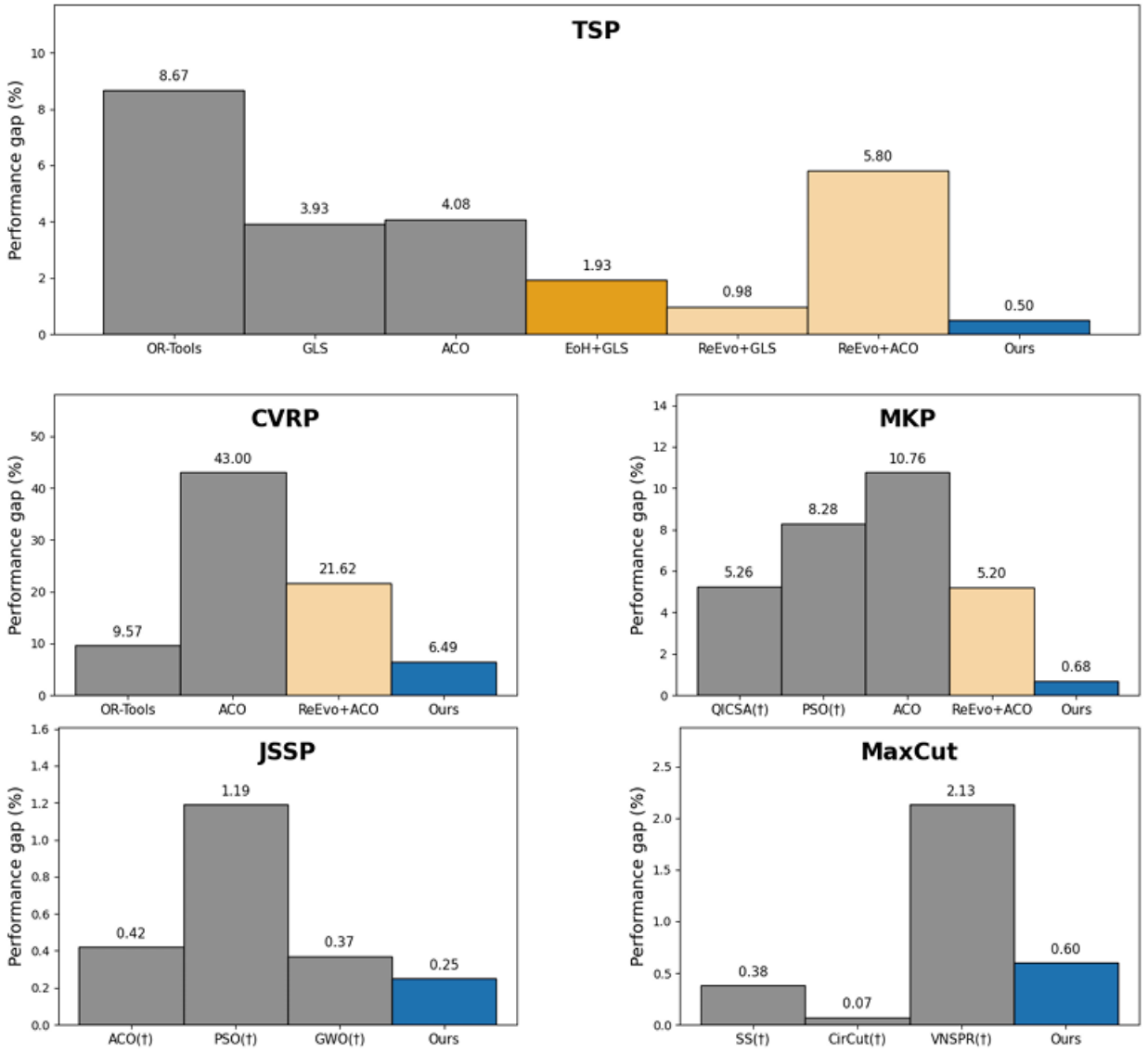
- ① 进化实验
  - 测试Heuristic Evolution能否发现更优的启发式



实验

- 背景知识
- 设计框架
- 实验
  - 进化实验
  - 求解实验
  - LLM微调性能实验
  - 蒙特卡洛搜索TTS实验
- Thinking

- ② 求解实验
  - 测试问题求解性能



实验

- 背景知识
- 设计框架
- 实验
  - 进化实验
  - 求解实验
  - LLM微调性能实验
  - 蒙特卡洛搜索TTS实验
- Thinking

③ 微调LLM求解实验

测试微调LLM性能

Table 3: Per-instance optimality gaps on TSPLIB (%; lower is better) when the selector is a mainstream LLM. **Bold** marks the best result. Variance is omitted when it equals 0.

Instance	GPT-4o	OpenAI O3	DeepSeek-R1	Ours
kroA100	**0**	**0**	**0**	**0**
kroA150	**0**	**0**	**0**	**0**
kroB100	**0**	**0**	**0**	**0**
kroB200	0.11	0.10 ± 0.1	**0**	0.30 ± 0.1
kroC100	**0**	**0**	**0**	**0**
bier127	1.29 ± 0.6	**0.22** ± 0.1	1.01 ± 0.1	1.09 ± 0.2
tsp225	0.23 ± 0.1	0.29 ± 0.1	**0.20**	0.24 ± 0.2
a280	0.16 ± 0.1	**0**	0.10 ± 0.1	0.91 ± 0.4
pcb442	2.10 ± 0.5	**0.59** ± 0.1	0.80 ± 0.5	0.79 ± 0.2
gr666	**0.93** ± 0.2	1.50 ± 0.3	1.19 ± 1.3	1.13 ± 0.9
pr152	0.23 ± 0.2	**0.13**	0.16	0.19 ± 0.1
pr1002	1.78 ± 0.6	1.33 ± 0.4	1.30 ± 0.3	**1.00** ± 0.3
pr2392	1.08 ± 0.5	**0.87** ± 0.3	1.09 ± 0.7	0.92 ± 0.3
Average gap	0.61	**0.39**	0.45	0.50

Table 4: Ablation studies based on Qwen-7B. Optimality gaps on TSPLIB (%; lower is better). **Bold** marks the best per row. Variance is omitted when it equals 0.

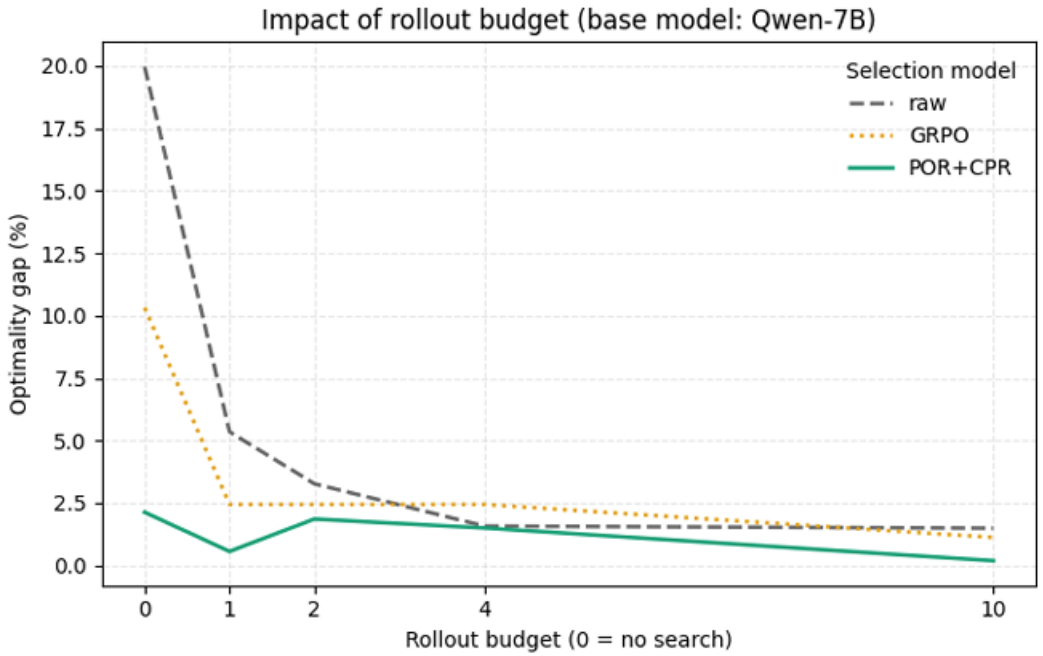
Instance	raw	GRPO	Ours
kroA100	**0**	0.21 ± 0.1	**0**
kroA150	0.80 ± 0.1	0.91 ± 0.1	**0**
kroB100	**0**	**0**	**0**
kroB200	2.82 ± 1.4	2.02 ± 1.1	**0.30** ± 0.1
kroC100	0.44 ± 0.1	0.10	**0**
bier127	2.82 ± 1.4	3.01 ± 1.1	**1.09** ± 0.2
tsp225	5.22 ± 3.4	4.01 ± 2.4	**0.24** ± 0.2
a280	4.82 ± 1.4	3.31 ± 0.4	**0.91** ± 0.4
pcb442	3.00 ± 1.9	2.21 ± 1.9	**0.79** ± 0.2
gr666	8.02 ± 5.4	4.82 ± 2.2	**1.13** ± 0.9
pr152	1.49 ± 0.4	1.12 ± 0.3	**0.19**± 0.1
pr1002	9.98 ± 8.4	5.02 ± 2.7	**1.00** ± 0.3
pr2392	10.86 ± 7.4	8.21 ± 7.3	**0.92** ± 0.3
Average gap	5.01	4.39	**0.59**

实验

- 背景知识
- 设计框架
- 实验
  - 进化实验
  - 求解实验
  - LLM微调性能实验
  - 蒙特卡洛搜索TTS实验
- Thinking

④ 蒙特卡洛搜索TTS实验

- 测试rollout budget对性能的影响



X轴：蒙特卡洛样本数 （rollout budget）
0表示不搜索，直接使用LLM选择的启发式

Thinking

- 能否进一步提高？
  - 1. 当前“离线进化”、“在线求解”为分开的两个阶段；可改进为：求解过程中若发现当前Heuristic Pool不足够好，则可以触发进化，实现持续学习
  - 2. “在线求解”部分中蒙特卡洛模拟的部分，本质上是求解一个选择的**未来预期收益**。当前方法的问题：方差大（随机策略）和计算成本高（需要模拟到完成）。参考RL，训练一个**Value Network**来直接**预测当前状态的预期收益**。
- 能否泛化？
  - 多目标优化场景：LLM分析trade-off，而不是“好坏”的差异
  - 思路：让LLM分析“为什么会变好”，而不是直接让LLM从旧方案生成新方案
- 能否用在我们的场景？
  - 混合精度量化