ReVEL: Multi-Turn Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback

Anonymous ACL submission

Abstract

Designing effective heuristics for NP-hard combinatorial optimization problems remains a challenging and expertise-intensive task. Existing applications of large language models (LLMs) primarily rely on one-shot code synthesis, yielding brittle heuristics that underutilize the models' capacity for iterative reasoning. We propose **ReVEL**: Multi-Turn **Re**flecti**v**e LLM-Guided Heuristic **E**vo**l**ution via Structured Performance Feedback, a hybrid framework that embeds LLMs as interactive, multi-turn reasoners within an evolutionary algorithm (EA). The core of REVEL lies in two mechanisms: (i) performance-profile grouping, which clusters candidate heuristics into behaviorally coherent groups to provide compact and informative feedback to the LLM; and (ii) multi-turn, feedback-driven reflection, through which the LLM analyzes group-level behaviors and generates targeted heuristic refinements. These refinements are selectively integrated and validated by an EA-based meta-controller that adaptively balances exploration and exploitation. Experiments on standard combinatorial optimization benchmarks show that REVEL consistently produces heuristics that are more robust and diverse, achieving statistically significant improvements over strong baselines. Our results highlight multi-turn reasoning with structured grouping as a principled paradigm for automated heuristic design.

1 Introduction

Large language models (LLMs) have recently demonstrated strong capabilities in tasks requiring multi-step reasoning, iterative refinement, and feedback integration, ranging from mathematical problem solving to program synthesis (Liao et al., 2026). These results suggest that LLMs can operate over intermediate states as structured reasoners, rather than merely producing one-shot outputs (Jiang et al., 2026).

A key open question is whether such reasoning capabilities can be effectively leveraged in settings where solution quality must be *iteratively improved under explicit performance feedback*. Combinatorial optimization problems (COPs) (Peres and Castelli, 2021; Martins et al., 2025) provide a canonical instance of this challenge: solutions are not evaluated once, but progressively refined within exponentially large discrete search spaces.

Representative problems such as the Traveling Salesman Problem (TSP) (Bock et al., 2025) and the Bin Packing Problem (BPP) (Aïder et al., 2025) are NP-hard and central to real-world applications in logistics, supply chain management, and transportation, where even marginal improvements yield significant operational gains. Due to the intractability of exact methods at scale, research has primarily focused on heuristic approaches, which can be broadly categorized by how heuristics are constructed and refined.

First, **classical heuristic and meta-heuristic methods** (Almufti and Shaban, 2025; Rondano, 2025) rely on manually designed operators that exploit problem-specific structures. Representative examples include local improvement rules, such as 2-opt and 3-opt (Qiao et al., 2025), for the TSP, and constructive strategies, such as First-Fit and Best-Fit, for the BPP (Rabelo, 2024). These methods are computationally efficient and widely used in practice, but their performance depends heavily on expert knowledge and fixed heuristics, which limit adaptability and long-term improvement.

Second, **hyper-heuristic approaches** (Xu et al., 2025; Vela et al., 2025; Marko et al., 2025) attempt to automate heuristic design by selecting or combining heuristics from a predefined pool. This reduces manual effort and improves generality, but the search remains constrained by the expressiveness of the available heuristic components. As a result, hyper-heuristics typically refine existing strategies rather than discovering fundamentally

new heuristic structures.

Third, recent advances in large-scale neural models have enabled **AI-driven program synthesis** (Hu, 2025; Shethiya, 2025). Systems such as AlphaCode (Li et al., 2022) show that Transformer-based models can synthesize non-trivial programs directly from problem descriptions. This capability suggests a new paradigm for COPs: instead of selecting predefined heuristics, models can generate novel ones. However, these approaches depend on large-scale sampling, with no principled mechanism to integrate performance feedback into the iterative improvement of generated heuristics.

Fourth, motivated by this capability, recent work has explored **LLM-driven evolutionary heuristic design** (Busch et al., 2025), integrating large language models into evolutionary optimization frameworks. In this paradigm, the LLM serves as a heuristic synthesis engine that generates candidate algorithms, while performance evaluation provides objective feedback. For example, *Evolution of Heuristics (EoH)* (Liu et al., 2024) co-evolves heuristic code together with reasoning traces, and *ReEvo* (Ye et al., 2024) introduces reflective critique to improve heuristic quality.

Despite promising results, existing approaches typically incorporate reflection as a *discrete step within a generation*, which limits the immediacy and depth of feedback integration. As a result, improvements in reasoning are delayed across generations, and the LLM receives only coarse-grained signals about heuristic performance.

These limitations suggest that the effectiveness of LLM-based heuristic discovery critically depends on how performance feedback is structured and delivered to the model.

Our approach. We propose REVEL, a hybrid evolutionary framework that integrates an evolutionary algorithm with a large language model (LLM) through reflective, multi-turn interaction. Rather than using the LLM as a static code generator, REVEL embeds it as an adaptive reasoning component that incrementally analyzes performance feedback, reflects on structural patterns across heuristics, and proposes targeted refinements after each evolutionary generation. Central to this design is a novelty-aware reflective mechanism, which balances exploration and exploitation by guiding the LLM to both introduce qualitatively new strategies and systematically improve promising heuristic structures.

Contributions. Our contributions can be summarized as follows:

- We introduce a reflective LLM–EA framework that enables multi-turn reasoning over accumulated evolutionary feedback, transforming heuristic discovery into a coherent refinement process rather than independent generation attempts.
- We develop a performance-aware grouping mechanism that structures reflective feedback around behaviorally coherent heuristic clusters, enabling more informative analysis and generalizable refinements.
- We design a feedback-driven, multi-turn prompting strategy integrated within an EA generation–evaluation–selection loop, enabling principled control over the exploration–exploitation trade-off.
- We empirically demonstrate, across benchmark combinatorial optimization problems, that REVEL consistently produces more robust and diverse heuristics, achieving statistically significant improvements over strong evolutionary and LLM-based baselines.

2 Related work

2.1 Automatic Heuristic Design

Traditional heuristic design relies heavily on domain-specific manual engineering, which is labor-intensive and hard to generalize (Burke et al., 2013). To automate this process, early hyper-heuristics employed Genetic Programming (GP) to evolve rules from primitive components (Branke et al., 2015). More recently, the focus has shifted to Neural Combinatorial Optimization (NCO), where Deep Reinforcement Learning (DRL) trains constructive policies directly on problem instances (Bello et al., 2017; Kool et al., 2018).

Despite their success, both GP-based and DRL-based approaches often require substantial training effort and remain sensitive to problem representations and distributional shifts. These limitations have motivated interest in more flexible heuristic synthesis paradigms that can leverage higher-level reasoning and reuse knowledge across tasks.

2.2 Large Language Models for Evolutionary Search

Large Language Models (LLMs) (Vaswani et al., 2017; Zhao et al., 2023) have recently emerged as

powerful tools for code generation and algorithmic search, owing to their ability to reason over program semantics rather than relying on purely syntactic perturbations. This capability enables LLMs to participate in multiple stages of the evolutionary loop, including candidate generation, variation, and selection. Compared to earlier code generation systems such as AlphaCode (Li et al., 2022), which primarily depend on massive sampling and post-hoc filtering, LLM-based frameworks emphasize iterative refinement guided by feedback from evaluators.

FunSearch (Romera-Paredes et al., 2023) pioneered the integration of LLMs into Evolutionary Computation (EC) by coupling a language model with a deterministic evaluator to iteratively evolve programs for combinatorial optimization. Subsequent works explored tighter LLM–EC coupling: EoH (Liu et al., 2024) co-evolves executable heuristics and natural-language "thoughts" via in-context learning, while LLaMEA (van Stein and Bäck, 2025) replaces standard genetic operators with LLM-based crossover and mutation to enable semantic recombination. Despite these advancements, such frameworks predominantly prioritize generation and variation, often overlooking the potential of structured critique to analyze failure modes and guide the search.

2.3 Multi-Turn Reflection

Beyond standard evolutionary search, an emerging line of work investigates multi-turn interaction and reflection mechanisms to improve LLM outputs. General-purpose frameworks like Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) demonstrate that LLMs can act as autonomous agents that critique and improve their own responses through iterative feedback loops. In the specific domain of code generation, Self-Debugging (Chen et al., 2024) extends this capability by enabling models to interpret execution feedback—such as unit test results or runtime traces—to identify and rectify implementation errors without human intervention.

In the context of heuristic optimization, ReEvo (Ye et al., 2024) adapted this reflective paradigm, establishing a strong baseline by prompting LLMs to mimic human debugging. ReEvo generates verbal critiques based on performance differences between heuristic pairs, subsequently refining the code. While this approach improves upon one-shot synthesis, its reliance on pairwise comparison restricts the diversity and richness of the feedback signal. Other recent methods, such as HSEvo (Dat et al., 2025), have attempted to address population diversity; yet, they often lack a structured mechanism for analyzing collective failure modes.

3 Methodology

3.1 Overview

Our proposed method, **Multi-Turn Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback (ReVEL)**, as shown in **Figure 1**, is motivated by the way human cognition develops strategies through incremental reasoning and self-reflection. Humans rarely arrive at effective solutions in a single attempt; instead, they progress step by step, comparing alternatives, grouping related ideas, reflecting on their strengths and weaknesses, and refining them through iterative deliberation. Inspired by this process, ReVEL establishes a multi-turn mechanism that enables large language models (LLMs) to evolve heuristics dynamically, transcending the limitations of one-shot generation.

The framework unfolds through stages:

- **Grouping and Behavioral Clustering:** The initial pool of heuristics is organized into groups using two complementary strategies: one ensures the existence of groups with structural and performance similarity, while the other enforces diversity in semantics and design. This dual mechanism establishes a rich basis for reflection, ensuring that feedback is contextualized both within coherent heuristic families and across contrasting perspectives.

- **Reflective Multi-turn Refinement:** Within each group, the LLM engages in a structured multi-turn dialogue. Instead of isolated prompt calls, the model critiques why certain heuristics succeed or fail, extracts comparative insights, and synthesizes refined or entirely novel variants. This reflective evolution mirrors human cognitive processes of critique and incremental improvement, allowing the search to progressively deepen.

3.2 Grouping and Behavioral Clustering

Representation. Each heuristic h is evaluated on a benchmark set $\mathcal{I} = \{1, \ldots, m\}$ with objective

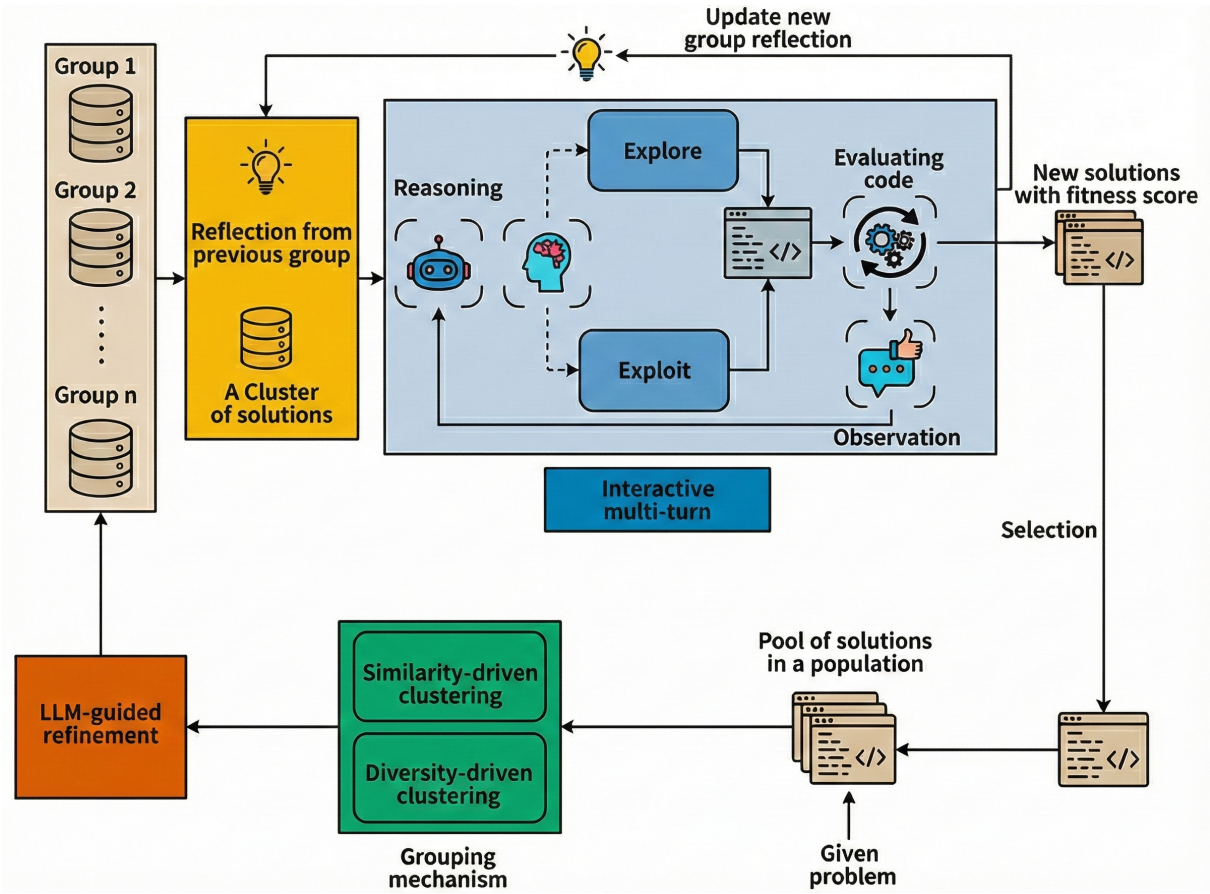


Figure 1: **Multi-Turn Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback (ReVEL):** The framework begins with Population Initialization to create a diverse set of candidate heuristics. Candidates are then organized in Grouping to ensure both coherence and diversity. Reflective Multi-turn Refinement iteratively evaluates, diagnoses, and improves heuristics. Finally, Population Management retains candidates that balance quality and diversity, enabling sustained heuristic evolution.

values $e_i(h)$ on instance i, and represented by a normalized performance-profile vector:

$$\mathbf{z}(h) = \frac{\mathbf{e}(h) - \mathbf{e}^*}{\mathbf{e}^*}, \tag{1}$$

in which,

$$\mathbf{e}(h) = \big(e_1(h), \ldots, e_m(h)\big)^\top, \tag{2}$$

$$\mathbf{e}^* = \big(e_1^*, \ldots, e_m^*\big)^\top, \tag{3}$$

$$e_i^* = \min_{h' \in \mathcal{H}} e_i(h'), \qquad i = 1, \ldots, m. \tag{4}$$

Here, e_i^* denotes the best-known cost on instance i across all heuristics in $\mathcal{H}$.

General idea. Adequate reflection arises only when the LLM interacts with *well-structured heuristic sets*. Homogeneous groups support fine-grained, instance-level comparison, whereas heterogeneous groups broaden the abstraction space and stimulate creative synthesis. ReVEL exploits both perspectives: it first constructs **homogeneous groups** using a static clustering method followed by LLM-based refinement, ensuring internally coherent sets of heuristics. It then forms **heterogeneous groups** by combining heuristics from distinct homogeneous groups, thereby increasing diversity and expanding the reflective space to elicit more innovative heuristic improvements.

Homogeneous groups (similarity-driven)

Homogeneous groups comprise heuristics with similar behavior or structural traits. We quantify similarity between heuristics h_i and h_j by integrating two complementary signals:

- **Performance similarity.** Behavioral similarity is measured in the normalized performance space $\mathbf{z}(\cdot)$ via cosine similarity:

$$\mathrm{sim}_{\mathrm{perf}}(h_i, h_j) = \frac{\mathbf{z}(h_i)^\top \mathbf{z}(h_j)}{\|\mathbf{z}(h_i)\|_2 \, \|\mathbf{z}(h_j)\|_2}, \tag{5}$$

- **Semantic similarity.** Structural resemblance is assessed with CodeBLEU, which extends BLEU by incorporating lexical, syntactic, semantic, and data-flow aspects:

$$\mathrm{sim}_{\text{code}}(h_i, h_j) \;=\; \mathrm{CodeBLEU}(h_i, h_j). \tag{6}$$

The overall similarity is defined as a weighted combination of the two signals:

$$\mathrm{sim}(h_i, h_j) = \alpha \mathrm{sim}_{\text{perf}}(h_i, h_j) + \beta \mathrm{sim}_{\text{code}}(h_i, h_j) \tag{7}$$

which $\alpha, \beta \geq 0$.

Clustering procedure:

To construct homogeneous groups we represent heuristics as a weighted similarity graph

$$G = (H, E, W), W_{ij} \;= \mathrm{sim}(h_i, h_j). \tag{8}$$

$$\begin{aligned} \tilde{W} &= \frac{W}{\max_{p,q} W_{pq}}, \\ d_{ij} &= 1 - \tilde{W}_{ij}, \quad D = (d_{ij})_{i,j}. \end{aligned} \tag{9}$$

where H is the set of heuristics and $W \in \mathbb{R}^{|H| \times |H|}$ stores pairwise similarities. Normalize W by its global maximum and form dissimilarities

Clustering is sensitive to granularity: too few groups collapses distinctions, while too many fragments structure. REVEL therefore uses a two-phase routine:

1. **Initial over-partitioning.** Apply agglomerative clustering with linkage L to obtain a fine partition

$$(C_1, \ldots, C_m) = \mathrm{Agglomerative}_L(D;\ m) \tag{10}$$

which $m \gg 1$
chosen so that clusters are tight:

$$\mathrm{diam}(C_\ell) = \max_{x,y \in C_\ell} d(x, y) \leq \delta, \qquad \delta \ll 1. \tag{11}$$

2. **LLM-guided refinement.** Provide the full over-partition $\mathcal{C}_0 = \{C_1, \ldots, C_m\}$ to an LLM which returns a globally restructured partition

$$\mathcal{C}^{\text{ref}} = \Phi_{\text{LLM}}(\mathcal{C}_0, W), \tag{12}$$

subject to the partition constraints

$$\bigcup_{C \in \mathcal{C}^{\text{ref}}} C = H, \qquad C_p \cap C_q = \emptyset \quad (p \neq q). \tag{13}$$

Heterogeneous groups (diversity-driven)

Diversity via entropy. The diversity of a homogeneous cluster $G = \{h_1, \ldots, h_m\}$ is quantified by the entropy of its internal similarity distribution. Since the composite similarity sim is symmetric, we consider only unordered pairs (i, j) with $i < j$. Normalized affinities are defined as

$$p_{ij} \;=\; \frac{\mathrm{sim}(h_i, h_j)}{\sum_{u<v} \mathrm{sim}(h_u, h_v)}, \qquad i < j, \tag{14}$$

The entropy score of G is then

$$\mathcal{H}(G) \;=\; -\sum_{i<j} p_{ij} \log p_{ij}, \tag{15}$$

with larger values indicating higher internal heterogeneity.

Entropy-weighted sampling. Given k homogeneous clusters $\{G_1, \ldots, G_k\}$, sampling weights are assigned proportionally to their entropy values:

$$w_i \;=\; \frac{\mathcal{H}(G_i)}{\sum_{j=1}^{k} \mathcal{H}(G_j)}, \qquad i = 1, \ldots, k. \tag{16}$$

For a target heterogeneous group size L, the contribution of each cluster is

$$L_i \;=\; \lfloor w_i \cdot L \rfloor, \tag{17}$$

L_i heuristics are drawn uniformly at random from G_i. The final heterogeneous group is the union

$$G^{\text{het}} \;=\; \bigcup_{i=1}^{k} S_i, \qquad S_i \subseteq G_i,\ |S_i| = L_i. \tag{18}$$

3.3 Reflective Multi-turn Refinement

REVEL organizes the search as a sequence of iterative turns, each following the loop

$$\text{observe} \;\rightarrow\; \text{reason} \;\rightarrow\; \text{act}.$$

This cycle ensures that each LLM invocation contributes to a coherent refinement trajectory rather than producing isolated trials.

State and reflection. At turn t, the LLM is provided with a compact state representation

$$S_t = \{(h_j, \mathrm{diag}_j)_{j=1}^{m},\ \text{history}_{<t},\ \text{directions}\} \tag{19}$$

Let h_j denote candidate heuristics within a group with diagnostic features diag_j (cost and delta-improvement vectors). Reflecting on S_t exposes structural weaknesses, recurrent failure modes, and latent strengths, grounding subsequent reasoning in accumulated performance evidence.

Adaptive strategy: exploration vs. exploitation. Conditioned on the reflective analysis, the model decides between two complementary modes. *Exploration* is triggered when recent refinements plateau or converge to structurally similar outcomes, signaling entrapment in a local optimum. In this mode, the LLM proposes divergent heuristics, such as new operators or recombinations across clusters, to enlarge the search horizon. *Exploitation* is selected when promising candidates are detected. Here, the model performs targeted refinement, including parameter tuning, incremental patching, or structural polishing, to systematically transform partial successes into competitive solutions. This dynamic alternation between diversification and intensification is critical for avoiding stagnation while consolidating gains.

Action and observation. Once a strategy is chosen, the model outputs actionable artifacts—such as patches, new DSL entries, or modified code—accompanied by a short rationale and confidence estimate. With probability p, additional performance observations relative to the current best are injected into S_t, sharpening the reflective analysis and aligning the strategy with outcome-based evidence. A fixed maximum-turn budget further regulates the trajectory: early turns favor exploratory breadth, while later turns naturally prioritize exploitative depth.

Resulting workflow. The integration of reflection, adaptive strategy selection, and observation-driven feedback yields coherent refinement trajectories $\{h^{(1)}, h^{(2)}, \ldots, h^{(T)}\}$. This workflow achieves sample-efficient improvement, systematically escaping local optima while progressively optimizing promising directions—outcomes unattainable under naive re-prompting.

4 Experimental

4.1 Empirical Evaluation

For evaluation, we seek to answer the following question:

RQ1.[Effectiveness] How effectively does REVEL generate higher-quality solver code compared to baseline approaches and the effect across different LLM types?

RQ2.[Reasoning] How does REVEL behave across successive reasoning iterations, and how do these iterations influence code correctness and solution quality?

RQ3.[Ablation Study] How critical is each component of our method to overall performance?

Datasets. We evaluate REVEL on two canonical combinatorial optimization benchmarks: TSP and online BPP. TSP instances range from 10 to 200 nodes, while BPP capacities span 100 to 500.

Models. We employ DeepSeek V3 as the backbone for heuristic generation.

Baselines. We benchmark against competitive state-of-the-art methods, including EoH (Liu et al., 2024) and Re-evo (Ye et al., 2024), as well as several classical heuristic approaches.

Metrics. We evaluate performance via the *optimality gap*, defined as the relative deviation from the best-known value, averaged over three independent runs.

4.2 Results

RQ1. [Effectiveness] We compare our method with strong baselines on BPP and TSP, and evaluate its robustness across different LLM backbones.

Quantitative Evaluation. Across all bin-packing settings (Table 1), **ReVEL** consistently outperforms both classical heuristics and recent adaptive methods. Under tight capacity (C=100), ReVEL reduces the excess fraction to **2–3%**, compared to **4–5%** for First Fit and Best Fit. At medium capacity (C=300), ReVEL achieves below **0.3%** excess on long streams, while adaptive baselines such as ReEvo fluctuate between **0.2–0.6%**. Even at large capacities (C=500), where all methods converge, ReVEL maintains a measurable advantage. A similar trend appears in TSP benchmarks (Table 2), where ReVEL achieves consistently lower optimality gaps than state-of-the-art LLM-based approaches (EOH, ReEvo) across instance sizes $n \in [10, 200]$. These results highlight ReVEL's ability to generalize across problem scales and combinatorial structures.

Effect of LLM Backbone. To isolate whether performance stems from the LLM or from the ReVEL workflow itself, we evaluate ReVEL with a diverse set of models—`deepseek-v3-0324`, `kimi-k2-instruct`, `qwen3-coder`, `480b-35a`, and `glm-4.5` which shown in Table 4. (These models span instruction-tuned, code-centric, and large general-purpose LLM families, providing a representative capability spectrum.) Across all backbones, ReVEL remains highly stable: the best average gap is obtained with `deepseek-v3-0324` (**1.13**) and the worst with `kimi-k2-instruct` (**1.95**). The narrow performance band indicates that strong re-

Table 1: Online bin packing results. Fraction of excess bins to lower bound (lower is better).

Heuristic	Capacity = 100			Capacity = 300			Capacity = 500		
	1k	5k	10k	1k	5k	10k	1k	5k	10k
First Fit	5.32%	4.40%	4.44%	1.34%	0.93%	0.92%	0.25%	0.50%	0.50%
Best Fit	4.87%	4.08%	4.09%	1.19%	0.84%	0.86%	0.25%	0.50%	0.47%
EoH	3.03%	2.15%	**0.33%**	0.60%	0.63%	0.58%	0.25%	0.50%	0.47%
ReEvo	3.78%	**0.80%**	**0.33%**	1.04%	0.27%	0.19%	0.25%	0.50%	0.47%
ReVEL (ours)	**2.34%**	1.13%	0.59%	**0.30%**	**0.24%**	**0.16%**	**0.25%**	**0.50%**	**0.45%**

Table 2: Baseline OR-Tools Results for the Traveling Salesman Problem.

Heuristic	**TSP10 (%)**	**TSP20 (%)**	**TSP50 (%)**	**TSP100 (%)**	**TSP200 (%)**
EoH	3.52	9.33	10.24	11.39	15.58
ReEvo	4.22	6.74	11.63	**11.01**	15.58
ReVEL (ours)	**2.11**	**6.74**	**9.20**	12.64	**11.46**

sults arise *not* from the raw power of the underlying LLM, but from ReVEL's structured reasoning and refinement workflow. This independence from a specific backbone underscores ReVEL as a robust and general optimization framework.

Table 3: Coefficient Tuning for TSP50

α	**Gap(%)**
0.2	15.79
0.4	14.06
0.5	**9.20**
0.6	15.97
0.8	12.50

RQ2.[Reasoning] We investigate how REVEL's multi-turn reasoning shapes code correctness and solution quality. Using LLM-assisted classification, each refinement step is categorized as a *paradigm shift*, *heuristic modification*, or *hyperparameter tuning* (Table 6). The distribution shows a dominant reliance on heuristic modification (**89**) and hyperparameter tuning (**75**), with paradigm shifts occurring only rarely (**16**), indicating that REVEL quickly commits to exploitation after brief initial exploration.

Turn-by-turn analysis (Figure 2) further reveals a consistent pattern: exploration via paradigm shifts occurs mainly at the first and final iterations. In contrast, the middle iterations concentrate on fine-grained exploitation. This behavior highlights a key insight: LLM-based optimization naturally follows an *explore-then-exploit* reasoning trajectory, with focused mid-stage refinement driving most of the performance gains.

RQ3.[Ablation Study] To disentangle the contribution of each component in our method, we conduct a comprehensive ablation study.

Table 4: Online Bin Packing: Method Across Models

Heuristic	**Model**	**Gap(%)**
ReVEL	deepseek-v3-0324	1.13
ReVEL	kimi-k2-instruct	1.95
ReVEL	qwen3-coder-480b-35a	1.41
ReVEL	glm-4.5	1.20

Effect of Multi-turn Refinement and Solution Grouping. Table 5 reports the results on the TSP50 benchmark. The full model achieves a **9.20%** optimality gap, while ablating *either* multi-turn refinement or solution grouping degrades performance to **17.18%**, the worst gap among all variants. This consistent degradation highlights that our strong performance cannot be attributed to a single design choice. Instead, it is the *joint effect* of multi-turn reasoning and grouping—both essential for stabilizing the LLM's code generation and guiding it toward higher-quality optimization solutions.

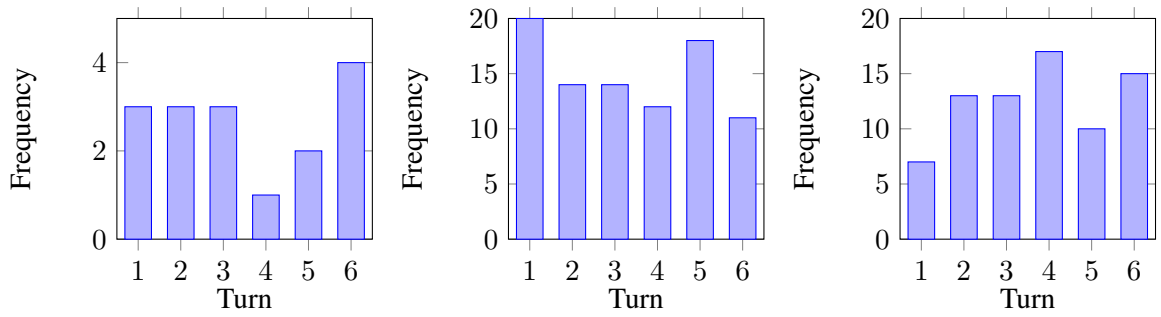
Table 5: Ablation study analyzing the impact of multi-turn and grouping mechanisms.

Setting	**Gap(%)**
w/o multi-turn	17.18
w/o grouping (random)	15.97
Ours (Full method)	**9.20**

Effect of Similarity Coefficient Tuning. We further investigate the impact of the parameter α, which balances behavioral similarity and

Table 6: Analysis of Reasoning Categories, their frequency, and examples.

Reasoning Category	Description	Freq	Example
Paradigm shift	Completely changes the algorithm family.	16	Given that the highest-performing known approach is "Stabilized Harmonic–Arctanh" . . . I should explore this proven algorithmic family rather than tuning simpler approaches.
Heuristic modification	Same algorithm family and pipeline structure, but decision and scoring logic is substantially rewritten.	89	Given the significant performance gap and clear indication that Worst-Fit works better, we should focus on refining this approach . . .
Hyperparameter tuning	Only numeric changes (weights, thresholds, constants) with formulas and pipeline unchanged.	75	Given the regression, we should revert to the simpler version and make minimal adjustments to core parameters . . .



(a) Paradigm Shift (b) Heuristic Modification (c) Hyperparameter Tuning

Figure 2: Frequency distributions of turns across three different scenarios.

CodeBLEU-based semantic similarity in our hybrid similarity metric. As shown in Table 3, the best performance is achieved at $\alpha = 0.5$, indicating that **both behavioral and semantic signals are indispensable**. This balanced configuration yields the most reliable similarity estimates, and we adopt $\alpha = 0.5$ across all experiments.

5 Conclusion

Novelty. REVEL introduces a reflective, multi-turn paradigm for LLM-guided heuristic evolution, reframing heuristic discovery as a coherent reasoning process rather than a sequence of independent code-generation attempts. The key novelty lies in embedding structured performance feedback into every model interaction, allowing the LLM to maintain a persistent reflective state that supports informed diagnosis, targeted refinement, and principled exploration–exploitation trade-offs. By organizing candidate heuristics into behaviorally coherent performance groups, REVEL provides a compact yet expressive context that enables deeper, more generalizable multi-turn reasoning within evolutionary search.

Implications. This formulation enables LLMs to function as adaptive problem solvers rather than static generators, yielding heuristics that are both more robust and more diverse. Beyond the specific benchmarks studied, REVEL establishes a general framework for integrating multi-turn reflective reasoning with evolutionary algorithms, suggesting a scalable and sample-efficient pathway for automated heuristic design in complex combinatorial optimization domains.

6 Limitations

Limitations. While structured performance feedback enables informed multi-turn reflection, it does not guarantee uniformly beneficial refinements. In some cases, group-level feedback may obscure fine-grained behavioral differences, leading the LLM to propose ineffective or redundant heuristic modifications. Although such proposals are filtered by evolutionary selection, this can reduce sample efficiency in certain stages of the search.

External Validity. The evaluated benchmarks do not capture the full diversity of real-world combinatorial optimization settings, though they represent widely studied and structurally distinct problem classes. To mitigate model-specific bias, REVEL was evaluated across multiple high-capacity LLMs—`deepseek-v3-0324`, `kimi-k2-instruct`, `qwen3-coder`, `480b-35a`, and `glm-4.5` (Table 4)—spanning instruction-tuned, code-oriented, and general-purpose architectures. Nevertheless, performance may vary under alternative model architectures or domain-specialized LLMs.

Internal Validity. The evolutionary process involves inherent stochasticity arising from initialization, variation operators, and LLM sampling. To control variance, all experiments were conducted with fixed random seeds, repeated across multiple independent runs, and evaluated using appropriate statistical tests.

Construct Validity. LLM behavior is sensitive to prompt formulation, and variations in lexical framing or structural emphasis may affect reflective reasoning. To ensure consistency, all prompts followed fixed templates with explicit structural constraints, as detailed in the Appendix.

References

Méziane Aïder, Sabrin Boulebene, and Mhand Hifi. 2025. An adaptative multi-objective scatter search for solving the dynamic bin packing problem. *Journal of Heuristics*, 31(1):5.

Saman M Almufti and Awaz Ahmed Shaban. 2025. Comparative analysis of metaheuristic algorithms for solving the travelling salesman problems'. *International Journal of Scientific World*, 11(2):26–30.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. *Preprint*, arXiv:1611.09940.

Stefan Bock, Stefan Bomsdorf, Nils Boysen, and Michael Schneider. 2025. A survey on the traveling salesman problem and its variants in a warehousing context. *European Journal of Operational Research*, 322(1):1–14.

Juergen Branke, Su Nguyen, Christoph Pickardt, and Mengjie Zhang. 2015. Automated design of production scheduling heuristics: A review. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,*, 20.

Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724.

Daniel Busch, Alexander Bainczyk, Steven Smyth, and Bernhard Steffen. 2025. Llm-based code generation and system migration in language-driven engineering. *International Journal on Software Tools for Technology Transfer*, 27(1):137–147.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. 2025. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 26931–26938. `https://github.com/datphamvn/HSEvo`.

Menghao Hu. 2025. Advancements and prospects in machine learning-driven code generation and completion.

Cong Jiang, Xiaofeng Zhang, Zheng Zhang, Fangzhi Zhu, XiaoWei Chen, and Junxiong Zhu. 2026. Rethinking LLM reasoning: From explicit trajectories to latent representations. In *The Fourteenth International Conference on Learning Representations*.

Wouter Kool, Herke van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Junfeng Liao, Qizhou Wang, Shanshan Ye, Xin Yu, Ling Chen, and Zhen Fang. 2026. Explainable LLM unlearning through reasoning. In *The Fourteenth International Conference on Learning Representations*.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *Forty-first International Conference on Machine Learning*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Marko, Francisco Javier Gil Gala, Domagoj Jakobović, and Yi Mei. 2025. Genetic programming as a hyper-heuristic for solving combinatorial optimisation problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, page 1070–1094, New York, NY, USA. Association for Computing Machinery.

Miguel SE Martins, João MC Sousa, and Susana Vieira. 2025. A systematic review on reinforcement learning for industrial combinatorial optimization problems. *Applied Sciences*, 15(3):1211.

Fernando Peres and Mauro Castelli. 2021. Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Applied sciences*, 11(14):6449.

Wen-Bao Qiao, Jean-Charles Créput, Nan Wang, Kun Meng, and Abdelkhalekn Mansouri. 2025. Breaking tsp local search barriers: Scalable multi-gpu parallelisation of 4-opt, 5-opt, 6-opt and hybrid variable λ-opt. *Expert Systems with Applications*, page 129110.

Ángel Martín Ramírez Rabelo. 2024. Exploration and analysis of heuristics and meta-heuristics for solving bin packing problem (bpp).

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Kumar, Emilien Dupont, Francisco Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2023. Mathematical discoveries from program search with large language models. *Nature*, 625.

Federico Rondano. 2025. *Heuristic Algorithms for the Min-Max Traveling Salesman Problem*. Ph.D. thesis, Politecnico di Torino.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Aditya S Shethiya. 2025. Ai-assisted code generation and optimization in. net web development. *Annals of Applied Sciences*, 6(1).

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Niki van Stein and Thomas Bäck. 2025. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *Preprint*, arXiv:2405.20132.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alonso Vela, Gerardo Humberto Valencia-Rivera, Jorge M Cruz-Duarte, José Carlos Ortiz-Bayliss, and Ivan Amaya. 2025. Hyper-heuristics and scheduling problems: strategies, application areas, and performance metrics. *IEEE Access*.

Song Xu, Xiangyue Ou, Kannan Govindan, Mingzhou Chen, and Wenting Yang. 2025. An adaptive genetic hyper-heuristic algorithm for a two-echelon vehicle routing problem with dual-customer satisfaction in community group-buying. *Transportation Research Part E: Logistics and Transportation Review*, 194:103874.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A Appendix

A.1 Preliminary

A.1.1 Evolutionary Algorithms

Formally, let $\mathcal{H}$ denote the space of all feasible heuristics. The goal of an Evolutionary Algorithm (EA) is to identify an optimal heuristic $h^* \in \mathcal{H}$ that maximizes a fitness function $\mathcal{F} : \mathcal{H} \to \mathbb{R}$:

$$h^* = \arg\max_{h \in \mathcal{H}} \mathcal{F}(h) \tag{20}$$

The algorithm maintains a population state at generation t, denoted as $\mathcal{P}^{(t)} = \{h_1^{(t)}, h_2^{(t)}, \ldots, h_N^{(t)}\}$, where N is the population size. The evolutionary process describes the transition $\mathcal{P}^{(t)} \to \mathcal{P}^{(t+1)}$ via a stochastic mapping $\mathcal{T}$:

$$\mathcal{P}^{(t+1)} = \mathcal{T}(\mathcal{P}^{(t)}; \mathcal{S}, \mathcal{V}) \tag{21}$$

where the transition relies on two primary operators:

1. **Fitness Evaluation:** Every individual $h_i^{(t)}$ is evaluated to obtain a quality score $s_i = \mathcal{F}(h_i^{(t)})$. This score typically drives the selection pressure.

2. **Selection Operator ($\mathcal{S}$):** This operator selects a subset of parents $\mathcal{P}_{parents} \subset \mathcal{P}^{(t)}$ based on their fitness. A common formulation is tournament selection, where a parent is chosen via:

$$h_{parent} = \arg\max_{h \in \mathcal{T}_{sub}} \mathcal{F}(h) \tag{22}$$

$$\mathcal{T}_{sub} \sim \text{Uniform}(\mathcal{P}^{(t)}, k) \tag{23}$$

   where $\mathcal{T}_{sub}$ is a random subset of size k.

3. **Variation Operator ($\mathcal{V}$):** In standard EAs, this involves Crossover ($\mathcal{C} : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$) and Mutation ($\mathcal{M} : \mathcal{H} \to \mathcal{H}$).

   In the context of **LLM-driven optimization**, the variation operator is re-parameterized as a conditional generation task using a Large Language Model π_θ. Given a prompt template I and parent heuristics, the offspring h_{new} is sampled from the model's distribution:

$$h_{new} \sim \pi_\theta(\cdot \mid I, \mathcal{P}_{parents}) \tag{24}$$

   Here, the LLM acts as a semantic mutation operator, leveraging its pre-trained knowledge to propose meaningful code transformations rather than random bit-flips.

Finally, the next generation $\mathcal{P}^{(t+1)}$ is formed by selecting the best candidates from the union of the current population and the newly generated offspring (elitism), ensuring monotonic improvement in the best-found solution over time.

A.1.2 From Pairwise to Group-wise Estimation

A central challenge in guiding LLM-based evolution is effectively estimating the relative quality of generated solutions to provide constructive feedback. Our methodology is directly motivated by the recent shift in preference optimization—specifically, the transition from pairwise comparisons to group-relative evaluations.

Limitations of Pairwise Comparison (DPO): Direct Preference Optimization (DPO) (Rafailov et al., 2023) aligns models by maximizing the likelihood margin between a winning sample y_w and a losing sample y_l. While effective for general alignment, this pairwise paradigm is suboptimal for evolutionary search. Evaluating a population of size N via pairwise comparisons restricts the model's view to local differences, ignoring the global context of how a solution stands against the entire population distribution.

The Group-wise Shift (GRPO): Group Relative Policy Optimization (GRPO) (Shao et al., 2024) addresses this by introducing a *group-wise* perspective. Instead of isolating pairs, GRPO evaluates a candidate o_i relative to a group of outputs $\{o_1, \ldots, o_G\}$ generated from the same prompt, using the group mean as a dynamic baseline:

$$A_i = \frac{f(o_i) - \text{mean}(\{f(o_j)\}_{j=1}^G)}{\text{std}(\{f(o_j)\}_{j=1}^G) + \epsilon} \tag{25}$$

Adoption in Our Approach: Inspired by this specific advancement, we adopt the *grouping mechanism* of GRPO as the core principle for our reflection module. We hypothesize that for an LLM to effectively critique and improve heuristics, it must analyze solutions not in isolation or pairs, but as a cluster. By referencing the statistical properties of the group (as proposed in GRPO), our system can identify relative outliers and extract more robust insights for the next evolutionary step.

A.2 Prompt

Online Bin Packing Prompt Formulation

```
TASK SUMMARY

You are an AI assistant whose job is to
     iteratively produce and refine
     Python heuristic implementations for
     the Bin Packing Online Problem.
You will be given an existing heuristic
     (or helper functions). Use
     multi-turn reasoning: at each turn
     you must reflect, then either
     **explore** a new heuristic family
     or **exploit** (refine) the last
     submitted heuristic, and finally
     receive an observation/feedback from
     the environment.

---

### FUNCTION CONTRACT (must be strictly
     respected)
- Language: Python only. Only standard
     library and numpy allowed (if
     already used by provided code).
- Required signature:
    def score(item, bins)
- Input arguments:
    - item: int # size of current item
    - bins : Numpy arrays # the rest
         capacities of feasible bins,
         which are larger than the item
         size.
- Return: scores (Numpy array)
- Correctness rules:
    - 'item' is of type int
    - 'bins' and 'scores' are both Numpy
         arrays.
```

Travelling Salesman Problem Prompt Formulation

```
TASK SUMMARY

You are an AI assistant whose job is to
     iteratively produce and refine
     Python heuristic implementations for
     the Travelling salesman problem.
Given a set of nodes with their
     coordinates, \
you need to find the shortest route that
     visits each node once and returns to
     the starting node. \
The task can be solved step-by-step by
     starting from the current node and
     iteratively choosing the next node. \
You will be given an existing heuristic,
     Let use multi-turn reasoning: at
     each turn you must reflect, then
     either **explore** a new heuristic
     family or **exploit** the last
     submitted heuristic, and finally
     receive an observation/feedback from
     the environment.

### FUNCTION CONTRACT (must be strictly
     respected)
- Language: Python only. Only standard
     library and numpy allowed (if
     already used by provided code).
- Required signature:
    def select_next_node(current_node,
         destination_node,
         unvisited_nodes, distance_matrix)
- Input arguments: This function should
     accept 4 inputs: 'current_node',
     'destination_node',
     'unvisited_nodes', 'distance_matrix'
- Output: The function should return 1
     output: 'next_node'
- Correctness rules: 'current_node',
     'destination_node', 'next_node', and
     'unvisited_nodes' are node IDs.
     'distance_matrix' is the distance
     matrix of nodes. All are Numpy
     arrays.
Do not give additional explanations.
```

Prompt Template for the THINK Step

```
ALWAYS REMEMBER THAT, LOWER fitness
     score = BETTER solution.
First, based on the evaluation result
     from <observation> or GROUP
     REFLECTION,
you should do some critical reasoning
     about the previous approach(s) ABOUT:
+ its logical algorithm
+ its heuristic
     components/hyperparamters/features
     specifically. Then, think about the
     affect of these
     parameters/hyperparamters to the
     fitness score result (in detailed).

Then, you can:

1. Explore a totally new approach, to
     make some experiments to get
     information.
OR
2. Focus on the behaviour of the
     heuristic features/components from
     the fitness result to tune them and
     get better result from the test
     evaluation.

You are ONLY allowed to do reasoning,
     NOT to generate code.
Note that, your reasoning should be very
     BRIEF but STILL critical and
     concise, focus on analyzing the
     heuristic components/features.
At the last of your response, there must
     be one of the tags <explore> or
     <exploit>, which indicate your
     decision.
```

Prompt Template for the Exploration Phase

```
Now, BASED solely on your REASONING,
    generate EXACTLY ONE solution for
    exploring.
Your output MUST be exactly the SAME as
    the following format:
<explore>
<algorithm>
# clear and complete algorithm
    description of the proposed
    heuristic.
</algorithm>
<code>
# the completely new Python function
    implementation for the algorithm in
    <algorithm> : `score(...)` (only
    code inside `<code>`).
</code>
</explore>
OUTPUT RULE:
Always output exactly one <explore>
    block containing both <algorithm>
    and <code>, nothing else.
```

Prompt Template for the Exploitation Phase

```
Now, BASED solely on your REASONING,
    generate EXACTLY ONE solution for
    exploiting.
Your output MUST be exactly the SAME as
    the following format:
<exploit>
<algorithm>
# Clear algorithm description of the
    improvements you're making to the
    selected algorithm
</algorithm>
<code>
# Complete and concise Python function
    implementation with your
    refinements: `score(...)`
</code>
</exploit>
OUTPUT RULE:
Always output exactly one <exploit>
    block containing both <algorithm>
    and <code>, nothing else.
```

LLM Prompt Template for Heuristic Population Clustering

```
You are given a list of candidate
    solutions for a heuristic problem.
Each candidate is a JSON object with
    fields:
- "id": unique identifier
- "code": the code itself
- "score": numeric performance value

Task: Group candidates that likely have
    the same behavior
1. Compare candidates by, code
    structure, and score values.
2. Group candidates into clusters of
    similar behavior, idea, or
    performance.
3. For each group, output:
  - Group label (integer ID).
  - List of member IDs.
  - Shared characteristics that define
      this group (algorithm style,
      coding pattern, performance
      range).
  - Distinctive differences within the
      group (if any).
4. If a candidate does not fit any
    existing group, assign it as its own
    group.
Also consider performance similarity:
- Candidates with very different scores
    (e.g., difference > 0.1 on a [01]
    scale or > 10% relative difference)
    should not be grouped, even if code
    is similar.
- Candidates with similar algorithmic
    ideas and close scores should be
    grouped together.

Output format:
Return ONLY a JSON object:

### example
{
  "groups": [
    {
      "id": 1,
      "members": ["1", "7", "9"],
      "shared_characteristics": "...",
      "differences": "...",
      "score_range": {"min": 0.72, "max":
          0.75}
    },
    {
      "id": 2,
      "members": ["3", "4"],
      "shared_characteristics": "...",
      "differences": "...",
      "score_range": {"min": 0.55, "max":
          0.58}
    },
    ...
  ]
}
###

CANDIDATE SOLUTIONS:\
```

A.3 Reasoning Behavior

As shown in Fig. 3, the Reflective LLM-Guided Heuristic Evolution via Structured Performance Feedback process demonstrates how score dynamics and strategic adaptation interact to drive progress beyond local optima. At the outset, when the system observed a fitness of 0.042, considerably worse than the best-known score of 0.031, it

recognized stagnation and initiated an exploratory shift. This led to the generation of Adaptive Resonance Packing (ARP), a structurally novel heuristic that improved the score to 0.036. Although this gain was modest, the introduction of ARP provided a fertile ground for subsequent refinements.

Building on this foundation, the process transitioned into targeted exploitation. By tuning ARP through dynamic bandwidth control, exponential gap weighting, and softmax normalization, the system sought to consolidate and optimize the idea. While the score did not improve beyond 0.036, this stage illustrates the reflective nature of the method: rather than abandoning a promising approach, it strategically invested in fine-tuning, ensuring stability before pursuing further change.

When performance remained unchanged, the system's observation mechanism signaled diminishing returns and prompted a more dramatic innovation. This shift produced Quantum Tunneling Bin Packing (QTBP), a probabilistic mechanism inspired by tunneling to bypass local barriers. Crucially, this step reduced the score to 0.023, surpassing the best-known baseline of 0.031.

Our approach integrates exploration, exploitation, and reflective reasoning: exploration opens new directions, exploitation strengthens promising structures, and reflection decides when radical change is needed. Unlike naive retries, it uses performance signals to avoid stagnation and achieve breakthroughs beyond conventional search.

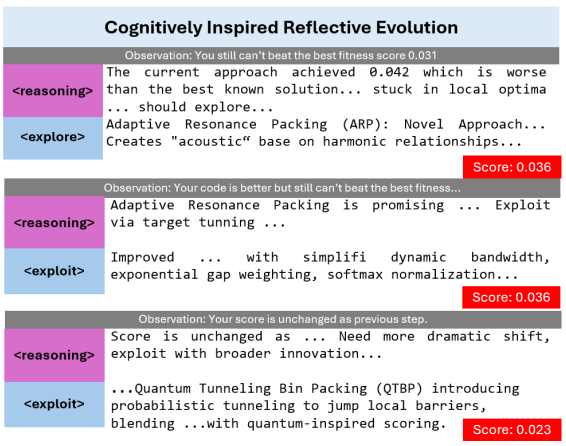


Figure 3: **Qualitative example of reflective reasoning in ReVEL:** Observations guide the LLM to alternate between exploration, exploitation, and innovation, resulting in progressive improvement of heuristic quality.

B Detail of benchmark dataset

B.1 Traveling Salesman Problem (TSP)

B.1.1 Problem Definition

The Traveling Salesman Problem (TSP) consists of finding the shortest possible tour that visits each city exactly once and returns to the starting city. Each instance is defined by a set of N cities with two-dimensional coordinates

$$\mathbf{x}_i = (x_i, y_i), \quad i = 1, \ldots, N,$$

where $x_i, y_i \in [0, 1]$.

The objective is to find a permutation π of $\{1, \ldots, N\}$ that minimizes the total tour length

$$L(\pi) = \sum_{k=1}^{N-1} d(\pi_k, \pi_{k+1}) + d(\pi_N, \pi_1).$$

B.1.2 Evaluation Metrics

The distance between cities i and j is the Euclidean distance $d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. We evaluate solution quality using the **Optimality Gap** relative to the OR-Tools baseline (L_{OR}):

$$\text{Gap } (\%) = \frac{L(\pi) - L_{OR}}{L_{OR}} \times 100\%$$

where $L(\pi)$ represents the total tour length. We report the average gap across all instances; a lower gap indicates higher proximity to the best-known solution.

B.1.3 Problem Sizes and Instances

We consider multiple TSP problem sizes in order to evaluate scalability across small, medium, and large instances. Table 7 summarizes all problem sizes used in our experiments.

Table 7: TSP problem sizes

Dataset	Number of Cities
TSP10	10
TSP20	20
TSP50	50
TSP100	100
TSP200	200

B.2 Online Bin Packing Problem

B.2.1 Problem Definition

In the online one-dimensional bin packing problem, items arrive sequentially. Each item must be

assigned immediately to a bin of C capacity. If no existing bin has sufficient remaining capacity, a new bin is opened.

Given a sequence of item sizes $s_1, s_2, \ldots, s_T$ with $s_i \in (0, 1]$, the objective is to minimize the total number of bins used. Reordering or moving previously packed items is not allowed.

B.2.2 Evaluation Metrics

For each instance, items are processed strictly in their arrival order (Online Bin Packing). Each item is either assigned to an existing bin with sufficient remaining capacity or placed into a newly opened bin. Once an item is placed, it cannot be moved or reassigned.

Let B denote the total number of bins used by an algorithm for a given instance I. We compare the algorithm's performance against the standard **volume lower bound**, defined as:

$$LB(I) = \left\lceil \frac{\sum_{i=1}^{T} s_i}{C} \right\rceil$$

where s_i is the size of item i, T is the sequence length, and C is the bin capacity.

The efficiency is measured using the **Fraction of Excess Bins**, expressed as a percentage:

$$\text{Fraction of excess bins}(I)(\%) = \left(\frac{B - LB(I)}{LB(I)}\right) \times 100\%$$

This metric quantifies the relative overhead compared to the theoretical minimum. A value of 0% indicates an optimal solution (matching the lower bound), while higher percentages reflect lower packing efficiency. Final results are reported as the average percentage across all test instances.

B.2.3 Problem Sizes and Instances

To evaluate performance across different scales, we construct datasets with varying bin capacities and number of items. Table 8 lists all configurations, where each dataset is defined by a pair (T, C) of number of items T and bin capacity C.

Table 8: Online Bin Packing problem sizes

Dataset	Number of Items (T)	Capacity (C)
BPP-1K-100	1K	100
BPP-1K-300	1K	300
BPP-1K-500	1K	500
BPP-5K-100	5K	100
BPP-5K-300	5K	300
BPP-5K-500	5K	500
BPP-10K-100	10K	100
BPP-10K-300	10K	300
BPP-10K-500	10K	500

C TSPLib

Evaluation on TSPLib Instances.

C.1 Evaluation on TSPLib Instances

To further evaluate the practical effectiveness and robustness of ReVEL, we conducted additional experiments on a diverse set of TSPLib benchmark instances. Unlike the synthetic Euclidean datasets used in previous sections, TSPLib instances exhibit heterogeneous spatial structures, irregular node distributions, and varying scales. They are widely regarded as representative of practical TSP scenarios.

For each TSPLib instance, ReVEL was executed independently under the same configuration to evolve a heuristic specialized for that instance. Table 9 reports the percentage gap to the best-known TSPLib solutions and compares ReVEL with Nearest Neighbour and EoH.

Overall, ReVEL achieves the best results on 12 out of 18 instances and consistently improves upon the Nearest Neighbour baseline, confirming that the discovered heuristics substantially outperform classical constructive strategies. Compared with EoH, ReVEL shows competitive performance across most instances and provides substantial improvements on several structurally diverse problems, including Berlin52 (6.24% vs. 17.68%), KroC100 (5.88% vs. 19.31%), Rat99 (12.30% vs. 21.07%), and Ts225 (3.73% vs. 8.99%).

Notably, the performance advantage of ReVEL is particularly pronounced on instances with irregular spatial structures, where purely local constructive rules are insufficient. This suggests that the structured reflective feedback in ReVEL enables the LLM to discover heuristics that capture higher-level structural patterns rather than relying solely on greedy proximity decisions.

Across different instance sizes, ReVEL maintains stable performance without instance-specific parameter tuning, indicating that the proposed reflective evolution mechanism scales consistently across problem regimes.

These results demonstrate that ReVEL can reliably guide heuristic discovery across heterogeneous instance structures and problem scales, extending its effectiveness beyond synthetic benchmarks to realistic combinatorial optimization problems.

Table 9: Results on TSPLib instances. The gap (%) to the best-known solution from TSPLib.

Instance	Nearest Neighbour	EoH	Our
Berlin52	19.06	17.68	**6.24**
Bier127	15.27	**8.04**	9.33
Ch130	23.98	**6.81**	10.21
D493	17.20	**14.85**	15.69
Eil51	19.72	**3.19**	3.68
KroA100	26.19	11.06	**6.21**
KroB100	31.68	9.84	**9.65**
KroC100	26.88	19.31	**5.88**
KroD100	26.56	13.21	**10.01**
KroE100	25.01	**11.79**	13.46
Pr124	16.97	16.69	**12.89**
Pr136	24.81	11.95	**6.73**
Pr226	18.52	11.66	**8.44**
Rat99	29.21	21.07	**12.30**
St70	19.33	16.33	**14.98**
Ts225	22.07	8.99	**3.73**
U159	32.93	11.92	**11.43**
Lin105	41.61	**7.53**	21.37

Table 10: Hyperparameters Configuration

Parameter Name	Value
Evolution Loop Parameters	
max_turns	6
num_candidates_to_initialize	10
epochs	20
top_k	10
reminder_probability	0.3
Clustering & Cross-over Parameters	
num_clusters	3
num_elements	4
alpha	0.5
groups_per_crossover	1
Evaluator Parameters	
timeout_seconds	70

Explanation of hyperparameters. As shown in Table 10, the hyperparameters are organized into three groups. The *Evolution Loop Parameters* control the overall evolutionary process, including the maximum number of interaction turns within each group (`max_turns`), the initial population size (`num_candidates_to_initialize`), the total number of evolutionary epochs (`epochs`), and the number of top-performing candidates preserved after each epoch (`top_k`). The parameter `reminder_probability` defines the probability of proactively injecting a reminder message to the LLM during the interaction loop, reinforcing awareness of the current best-fitness solution.

The *Clustering & Cross-over Parameters* determine how candidates are grouped and recombined. Specifically, `num_clusters` specifies the number of clusters formed from the population, `num_elements` controls how many representative candidates are selected from each cluster for crossover, and `alpha` balances semantic output similarity and code-level similarity during clustering. The parameter `groups_per_crossover` enforces that each cross-over operation produces exactly one new interaction group.

Finally, the *Evaluator Parameters* define execution constraints during evaluation. The parameter `timeout_seconds` limits the maximum runtime for evaluating each generated candidate, preventing non-terminating executions and ensuring stable evolutionary progress.

D Additional Baseline Results on CVRP

The Capacitated Vehicle Routing Problem (CVRP) is a classical combinatorial optimization problem in which a fleet of vehicles with limited capacity must serve a set of customers with known demands while minimizing the total travel cost. Each vehicle starts and ends at a depot, and customer demands must not exceed vehicle capacity.

To assess the robustness of **ReVEL** beyond TSP and BPP, we conducted additional experiments on the *CVRP* using the same experimental configuration as in the main paper. In this setting, heuristics were evolved independently for each problem size using an *ACO-based backbone*, allowing ReVEL to adapt to different combinatorial structures and heuristic paradigms.

For a strong and meaningful comparison, we evaluated against **ReEvo**, a recent state-of-the-art LLM-guided heuristic evolution framework for this task. Since both methods operate under similar LLM-driven evolutionary settings, this comparison provides a controlled evaluation of the reflective evolution mechanism.

In table 11 ReVEL consistently outperforms ReEvo across all tested scales, with particularly large improvements on **CVRP20 (4.57% vs. 13.70%)** and clear gains on larger instances as well. These results demonstrate that the proposed reflective evolution framework is **not limited to a specific problem type or heuristic structure**, but can effectively guide heuristic discovery under different optimization paradigms.

Importantly, the CVRP experiments use **the same default configuration without problem-specific tuning**, indicating that ReVEL generalizes across combinatorial domains with minimal adjustment.

Overall, these additional results provide strong empirical evidence that ReVEL is a **general LLM-guided heuristic evolution framework**, rather than a method specialized for a single problem class.

Table 11: Baseline OR-Tools Results for the Capacitated Vehicle Routing Problem.

Heuristic	CVRP20 (%)	CVRP50 (%)	CVRP100 (%)
ReEvo	13.70	19.44	18.06
ReVEL (ours)	**4.57**	**17.68**	**14.88**

E Cost–Performance Trade-off and Reflective Improvement Analysis

To better understand the efficiency and mechanism-level benefits of multi-turn reflection, we analyze both the computational cost and the performance trajectories of the generated heuristics. Figures 4 and **??** report the optimization trajectories, along with the accumulated evaluation cost, for representative instances of the Traveling Salesman Problem (TSP20 and TSP50) and the Bin Packing Problem (BPP300–2000 items).

The results demonstrate a clear advantage in the cost–performance trade-off achieved by multi-turn reflection. On TSP50, ReVEL achieves a solution gap of approximately 9% with a total cost of $0.68\$$. In contrast, single-turn reflection achieves only about 17% gap at $0.15\$$, and the no-reflection baseline attains approximately 18% gap at $0.37\$$. Similar behavior is observed on TSP20, where ReVEL consistently produces lower solution gaps while maintaining comparable computational cost.

The advantage of multi-turn reflection becomes even more pronounced in more challenging scenarios. On BPP instances with 300–2000 items, both single-turn reflection and the no-reflection baseline fail to generate any heuristic that improves upon the initial performance, even after the same number of evolutionary steps. In contrast, ReVEL continues to produce progressively improved heuristics under identical evaluation settings.

Although ReVEL requires a moderately higher computational cost, the improvement in solution quality is substantial. The additional cost mainly arises from two components:

- **Pre-generation reflection**, which enables the language model to interpret structured performance feedback before heuristic synthesis;
- **Multi-turn refinement**, which allows heuristics to be iteratively improved rather than generated independently.

These mechanisms enable more effective exploration of the heuristic space and improve the utilization efficiency of the evaluation budget.

The trajectory analysis also provides evidence that multi-turn reflection leads to progressive improvement of heuristic quality. Both single-turn reflection and the no-reflection baseline exhibit rapid early improvement, followed by clear performance plateaus, even when given the same computational

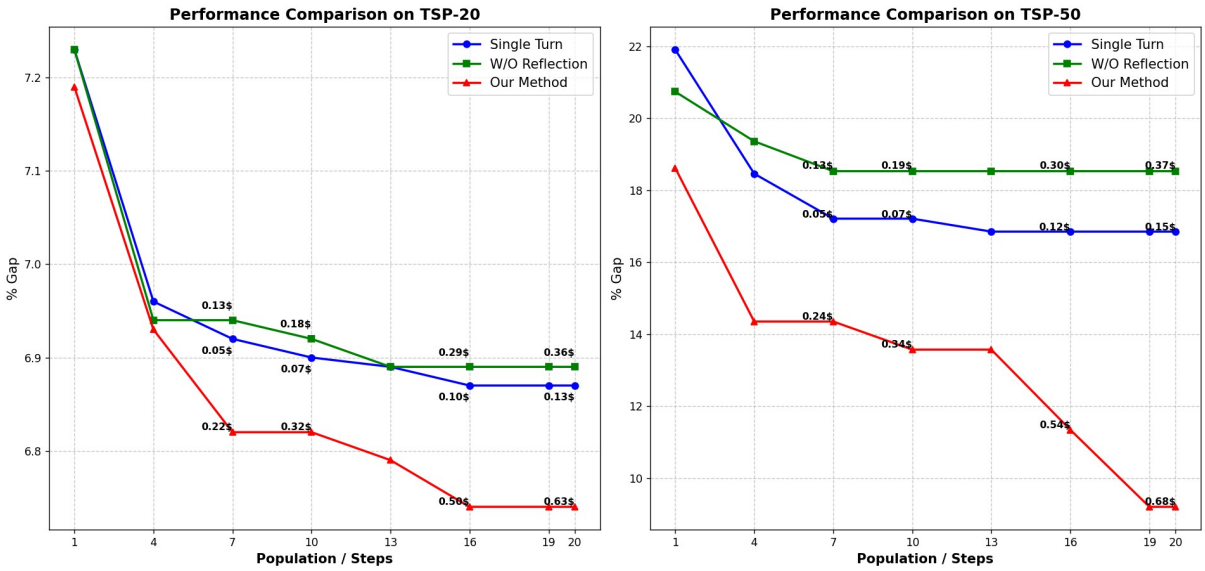


Figure 4: Performance comparison of Single Turn, W/O Reflection, and our method on TSP-20 and TSP-50.

budget as ReVEL. This behavior suggests limited reasoning depth and restricted capability to refine heuristic logic.

In contrast, ReVEL demonstrates steady and sustained improvement throughout the reflective process. As structured feedback accumulates across reflection turns, the language model can refine heuristic strategies consistently and coherently. This results in a sequence of heuristics with progressively improved performance rather than a collection of largely independent candidates.

Overall, the results indicate that multi-turn reflection is not merely an additional computational expense but a mechanism that enables structured iterative improvement of heuristic programs. The proposed framework achieves a favorable cost–performance trade-off by producing substantially better heuristics with only a modest increase in computation while providing clear empirical evidence of progressive heuristic refinement.

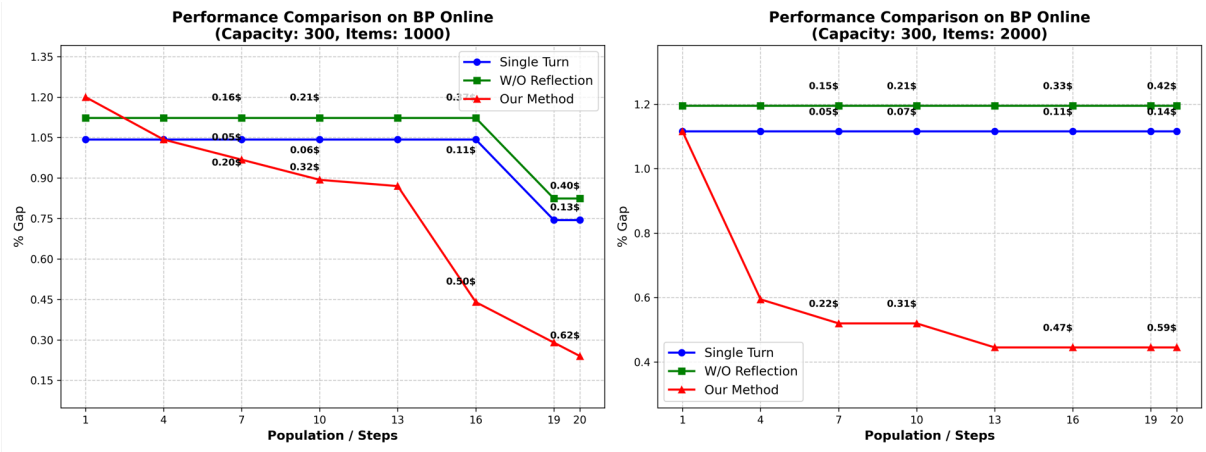


Figure 5: Performance comparison on online bin packing with capacity 300 and 1000/2000 items.

Algorithm 1 ReVEL: Multi-Turn Reflective Evolution

Require: Instance set $\mathcal{D}$, population size N, generations G, reflection turns T
Ensure: Best heuristic h^*

Initialize population $\mathcal{H}^{(0)} = \{h_1, \ldots, h_N\}$ using LLM
for $g = 1$ to G **do**
  Evaluate each heuristic $h \in \mathcal{H}^{(g-1)}$
  Compute fitness:

$$f(h) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \text{score}(h, x)$$

  Compute performance vectors $\mathbf{z}(h)$
  Build similarity clusters $\mathcal{C}$ using $\mathbf{z}(h)$
  Construct reflection groups $\mathcal{G} = \{G_1, \ldots, G_M\}$
  (each G_j mixes similar and heterogeneous heuristics)
  for each group $G_j \in \mathcal{G}$ **do**
    Initialize feedback $\mathcal{F}_j^{(1)}$ from:
    group performance statistics
    representative heuristics
    behavioral differences
    for $t = 1$ to T **do**
      Query LLM using feedback $\mathcal{F}_j^{(t)}$
      Generate candidate heuristics $\tilde{\mathcal{H}}_j^{(t)}$
      Evaluate candidates and update feedback:

$$\mathcal{F}_j^{(t+1)} = \mathcal{F}_j^{(t)} \cup \text{Perf}(\tilde{\mathcal{H}}_j^{(t)})$$

    end for
  end for
  Collect candidates:

$$\tilde{\mathcal{H}} = \bigcup_{j=1}^{M} \bigcup_{t=1}^{T} \tilde{\mathcal{H}}_j^{(t)}$$

  Select next population $\mathcal{H}^{(g)}$ from $\mathcal{H}^{(g-1)} \cup \tilde{\mathcal{H}}$
end for
return

$$h^* = \arg\min_h f(h)$$