PyVRP+: LLM-Driven Metacognitive Heuristic Evolution for Hybrid Genetic Search in Vehicle Routing Problems

Manuj Malik*
Singapore Management University
Singapore, Singapore
manujm@smu.edu.sg

Jianan Zhou*
Nanyang Technological University
Singapore, Singapore
jianan004@e.ntu.edu.sg

Shashank Reddy Chirra
University of Oxford
Oxford, England
shashank@robots.ox.ac.uk

Zhiguang Cao
Singapore Management University
Singapore, Singapore
zgcao@smu.edu.sg

ABSTRACT

Designing high-performing metaheuristics for NP-hard combinatorial optimization problems, such as the Vehicle Routing Problem (VRP), remains a significant challenge, often requiring extensive domain expertise and manual tuning. Recent advances have demonstrated the potential of large language models (LLMs) to automate this process through evolutionary search. However, existing methods are largely reactive, relying on immediate performance feedback to guide what are essentially black-box code mutations. Our work departs from this paradigm by introducing Metacognitive Evolutionary Programming (MEP), a framework that elevates the LLM to a strategic discovery agent. Instead of merely reacting to performance scores, MEP compels the LLM to engage in a structured Reason-Act-Reflect cycle, forcing it to explicitly diagnose failures, formulate design hypotheses, and implement solutions grounded in pre-supplied domain knowledge. By applying MEP to evolve core components of the state-of-the-art Hybrid Genetic Search (HGS) algorithm, we discover novel heuristics that significantly outperform the original baseline. By steering the LLM to reason strategically about the exploration-exploitation trade-off, our approach discovers more effective and efficient heuristics applicable across a wide spectrum of VRP variants. Our results show that MEP discovers heuristics that yield significant performance gains over the original HGS baseline, improving solution quality by up to 2.70% and reducing runtime by over 45% on challenging VRP variants.

KEYWORDS

Metaheuristic, Vehicle Routing Problem, Combinatorial Optimization, Large Language Model, Evolutionary Algorithm

ACM Reference Format:
Manuj Malik*, Jianan Zhou*, Shashank Reddy Chirra, and Zhiguang Cao. 2026. PyVRP+: LLM-Driven Metacognitive Heuristic Evolution for Hybrid Genetic Search in Vehicle Routing Problems. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 18 pages. https://doi.org/10.65109/

* Corresponding authors.

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/

1 INTRODUCTION

Vehicle routing problems (VRPs) represent the foundation of combinatorial optimization (CO) with profound implications for logistics, supply chain management, and transportation. Given their NP-hard nature, exact methods are intractable for large-scale, real-world instances, leading to the widespread adoption of metaheuristics. These high-level algorithmic frameworks are designed to efficiently explore the solution space for near-optimal solutions. Among these, Hybrid Genetic Search (HGS) has emerged as a State-Of-The-Art (SOTA) approach, demonstrating exceptional performance across numerous VRP variants [24]. Despite its effectiveness, HGS remains heavily dependent on human-crafted algorithmic components. Like many metaheuristics, its performance hinges on carefully tuned internal mechanisms, such as parent selection, survivor selection, and penalty adjustment, which are designed through tedious trial and error and require significant domain expertise.

Recently, Large Language Models (LLMs) have emerged as a promising tool for automatic heuristic design and scientific discovery [18]. By leveraging their ability to reason over code, data, and mathematical structure, LLMs have the potential to surpass human-engineered solutions across a wide range of domains. For instance, FunSearch [19] integrated LLMs into an evolutionary framework to discover novel solutions to mathematical problems, while AlphaEvolve [18] extended this paradigm to generate strategies that outperform human-designed algorithms in domains such as matrix multiplication, kernel engineering, and beyond. Beyond raw performance, LLM-driven evolution condenses the costly manual design cycle—often requiring weeks of domain expert effort—into a manageable automated process.

Similar progress has also been made in the domain of CO. Evolution of Heuristics (EoH) [13] demonstrated that LLMs can effectively automate heuristic design within an evolutionary loop by co-evolving solutions at two levels of abstraction: high-level thoughts expressed in natural language and their corresponding executable code. ReEvo [32] further leveraged "verbal gradients" derived from performance feedback to iteratively refine heuristics.

While promising, these approaches mainly focus on simple heuristics for standard CO problems such as the traveling salesman problem (TSP), and advancing traditional solvers beyond human capability remains a challenge. While these pioneering works successfully

demonstrate the potential of LLM-driven evolution, they often treat the LLM as a reactive code generator, relying predominantly on immediate performance feedback to steer the search. This paper argues that such an approach underutilizes the reasoning capabilities of modern LLMs. Our core contribution is therefore not only the use of an LLM for evolution but also the introduction of Metacognitive Evolutionary Programming (MEP), a framework that shifts the paradigm from reactive mutation to proactive, structured discovery. MEP elevates the LLM from a simple code mutator to a strategic discovery agent by embedding two key innovations: a *Domain-Aware Initialization* phase that grounds the model in established strategic principles, and a mandatory *Reason-Act-Reflect* cycle that compels the LLM to explicitly diagnose failures, formulate testable design hypotheses, and critically self-assess its own output. This metacognitive scaffolding is the primary driver of our results, enabling the discovery of heuristics that not only perform high but also advance SOTA in VRP solving with minimal human intervention.

By framing the evolutionary search as a hypothesis-driven discovery process, our aim is to show that heuristic evolution can systematically uncover novel heuristics that outperform their human-designed counterparts. Our work builds on the framework introduced by [1], which models human-like metacognitive reasoning. We structure our methodology in two key phases:

- **Domain-Aware Initialization**: We equip the LLM with domain knowledge via three critical components: common pitfalls (K_p) that undermine the target component, proven mitigation strategies (K_s) to address these pitfalls, and problem-specific traps (K_t) unique to VRP solving. This knowledge foundation primes the LLM with the necessary domain understanding before the evolutionary search begins.
- **Reason-Act-Reflect Cycle**: Each generation in the evolutionary loop of MEP follows a structured cognitive cycle. The LLM is prompted to: 1) *Reason*: Explicitly diagnose the shortcomings of parent heuristics. 2) *Act*: Formulate a clear design hypothesis and implement it as a new code-based heuristic. 3) *Reflect*: Assess its own generated code by articulating its rationale and potential limitations directly within the code's documentation.

We apply MEP to the challenging task of discovering high-performance components for the HGS algorithm, using the open-source PyVRP library [26] as our testbed. The discovered collection of high-performing heuristics, which we term PyVRP$^+$, demonstrates that a structured, hypothesis-driven evolution can produce novel HGS variants that advance the state-of-the-art in VRP solving. The results show that MEP is capable of evolving heuristics that deliver performance gains of up to **2.70%**, while simultaneously reducing runtime by over **45%** on challenging VRP variants. These improvements underscore MEP's potential as a compelling alternative to hand-crafted HGS components and represent an encouraging step toward automated algorithm discovery in CO.

2 RELATED WORK

Recent research has increasingly adopted LLMs to automate the design of heuristics for CO. Pioneering this direction, FunSearch [19] demonstrated that LLMs could effectively generate core heuristic components, circumventing the need for manual redesign of entire algorithms. Similarly, Liu et al. [12] employed LLMs to generate key evolutionary algorithm components, such as crossover and mutation operators. Further advancing this line of work, the Evolution of Heuristics (EoH) framework [13] introduced a method for iteratively refining heuristic seed functions through evolutionary computation. Most of these approaches have adopted evolutionary frameworks, integrating advanced techniques to enhance their efficacy. These techniques include reflective feedback mechanisms [32], predictive modeling for performance estimation [27], and strategies that encourage heuristic diversity [8]. This paradigm has successfully extended beyond single-objective scenarios, addressing multi-objective optimization problems [31] and diverse CO domains including SAT [21], MILP [33], and dynamic Job Shop Scheduling Problems (JSSP) [10]. Moving beyond single-agent approaches, recent studies [21, 29] have explored multi-agent frameworks, achieving enhanced performance through collaborative heuristic generation. To overcome limitations associated with population-based methods, Zheng et al. [35] introduced a Monte Carlo Tree Search (MCTS) strategy, integrating LLM-generated heuristics to boost exploration efficiency. More recent advancements focus on reducing manual intervention. Thach et al. [23] proposed a novel end-to-end framework based on problem reduction, which enables heuristic methods to function independently of predefined general algorithmic templates. Similarly, Shi et al. [20] developed a meta-optimization framework that systematically discovers robust optimizers for a range of heuristic design tasks, promoting broader exploration and operating at a higher abstraction level. Furthermore, Šurina et al. [22] suggested integrating evolutionary heuristic generation with reinforcement learning-based fine-tuning, shifting LLM usage from static heuristic generation towards dynamic refinement of both heuristics and the underlying models. However, these approaches primarily focus on evolving simple heuristics or individual components thereof, rather than tackling sophisticated and well-established traditional solvers. Most existing work targets constructive heuristics for basic problem variants or designs standalone neural policies that bypass traditional optimization entirely [2, 5]. Concurrently, LLMs have also been explored as end-to-end CO solvers [11], for refining HGS via RL-finetuned LLMs [37], and within agentic frameworks for complex VRPs [34]. In this paper, we bridge this gap by demonstrating that *LLM-guided evolution can enhance the performance of a SOTA solver like HGS across multiple challenging VRP variants, moving beyond proof-of-concept demonstrations to practical algorithmic improvements.* For a comprehensive review of the use of LLMs in CO, we direct readers to [7].

LLM-guided scientific discovery is rapidly expanding beyond combinatorial optimization. For instance, Chen et al. [6] apply LLMs to Neural Architecture Search, Goldie et al. [9] use them to discover novel loss functions and optimizers for reinforcement learning, and Nadimpalli et al. [16] explore their use in evolving activation functions. While most of these methods focus on evolving function-level programs, Novikov et al. [18] scale this approach to entire files with hundreds of lines of code, enabling the discovery of novel algorithms for matrix multiplication, solutions to open mathematical problems, kernel optimization, and more. Although these methods often outperform existing human-designed baselines within their domains, more ambitious efforts, such as AI Scientist [14, 28], aim

for fully autonomous end-to-end scientific discovery with minimal human intervention. While such systems demonstrate impressive capabilities, they still fall short of matching human-level performance (e.g., producing research papers that are accepted at top-tier conferences). To support progress in this direction, MLGym [17] introduces a suite of benchmark environments specifically designed for training and evaluating models on scientific discovery tasks, signaling a promising future for this emerging field.

3 PRELIMINARIES

This section presents the necessary background on Vehicle Routing Problems and the Hybrid Genetic Search algorithm.

3.1 Vehicle Routing Problem

The VRP is a class of combinatorial optimization problems focused on finding optimal sets of routes for a fleet of vehicles to serve a set of customers. A VRP is typically defined on a graph $G = (V, A)$, where V is a set of nodes and A is a set of arcs. The node set V is partitioned into a set of depots and a set of customers. Each arc $(i, j) \in A$ is associated with a travel cost $c_{i,j}$. The objective is to determine a set of vehicle routes, starting and ending at a depot, that visit all customers while minimizing a specific objective function, usually the total travel cost. To systematically handle the diverse landscape of VRP variants, we adopt the composable attribute framework used in [3, 4, 36]. This allows any variant to be defined by a combination of fundamental properties, facilitating structured analysis and evaluation. These attributes are categorized as follows:

- **Node-specific attributes**, which define local requirements for each customer or depot. The primary examples are:
  - **Demand**: The quantity of goods for pickup or delivery at a node i, denoted as q_i. This directly influences vehicle load throughout a route.
  - **Time Windows (TW)**: The required service interval $[a_i, b_i]$ for a customer, which imposes critical temporal constraints on the route schedule.
  - **Service Time**: The duration s_i required to complete service at a node, which consumes time and affects the feasibility of subsequent stops.
- **Global attributes**, which apply to entire routes or the problem as a whole. Key global constraints include:
  - **Capacity (C)**: The maximum load Q_k that a vehicle can carry, limiting the total demand of customers on a single route.
  - **Duration Limits (L)**: A maximum allowable travel distance or time for a route, ensuring operational constraints are met.
  - **Backhauling (B/MB)**: Rules governing routes with both deliveries (linehauls) and pickups (backhauls), often enforcing precedence constraints.
  - **Open Routes (O)**: A structural variant where vehicles are not required to return to the depot after their final delivery.
  - **Multi-Depot (MD)**: Scenarios involving multiple starting and ending points for vehicles, adding a layer of assignment complexity.

This unified taxonomy provides a clear and systematic method for representing complex VRPs, enabling the evaluation of algorithmic components across a wide spectrum of problem types.

3.2 VRP Variants Considered

In this work, we evaluate our approach on a diverse set of VRPs to test the generalization of the discovered heuristics. We consider the following 7 VRP variants:

- **Traveling Salesman Problem (TSP)**: The TSP represents the simplest case of the VRP family, involving a single vehicle (or salesman) that must visit every customer exactly once and return to the starting depot. Despite its conceptual simplicity, the TSP is NP-hard and serves as a building block for more complex routing problems. The objective is to minimize the total travel distance or cost of the tour.
- **Capacitated VRP (CVRP)**: The CVRP extends the basic VRP by introducing vehicle capacity constraints. Each vehicle k has a finite capacity Q_k, and each customer i has a demand q_i that must be satisfied. The constraint ensures that the total demand of all customers served by a single route cannot exceed the vehicle's capacity: $\sum_{i \in R_k} q_i \leq Q_k$, where R_k represents the set of customers served by vehicle k. This variant is fundamental in logistics applications where vehicles have physical limitations.
- **Generalized Vehicle Routing Problem (GVRP)**: In this variant, graph nodes are grouped into mutually exclusive clusters, and serving any single node within a cluster satisfies the service requirement for the entire cluster. This creates a combinatorial challenge where the algorithm must select both which clusters to visit and which specific node within each cluster to serve. The generalized structure is applicable to scenarios where customers have multiple service locations or where service points can substitute for each other.
- **Multi-Depot VRP with Time Windows (MDVRPTW)**: This variant combines the complexity of multiple depot locations with time window constraints. Each vehicle is based at one of several depots and must serve customers within their specified time windows $[a_i, b_i]$. The challenge lies in optimally assigning customers to depots and vehicles while respecting both spatial proximity and temporal feasibility. Each depot typically has its own fleet of vehicles, and routes must start and end at the same depot. This variant is particularly relevant for logistics companies with multiple distribution centers serving geographically dispersed customers with delivery time requirements.
- **Prize-Collecting VRP with Time Windows (PCVRPTW)**: Unlike traditional VRP variants where all customers must be visited, the PCVRPTW allows for selective customer service. Each customer i has an associated prize π_i collected when visited, and the objective is to maximize the total collected prize minus the total travel cost, subject to a total cost budget B. Customers have time windows, and the challenge is to identify the most profitable subset of customers to serve within the available resources.
- **VRP with Backhauls (VRPB)**: The VRPB addresses scenarios where customers are divided into two categories: linehaul customers (requiring delivery from the depot) and backhaul customers (requiring pickup to the depot). A critical constraint is that all linehaul customers must be served before any backhaul customers on the same route. This precedence

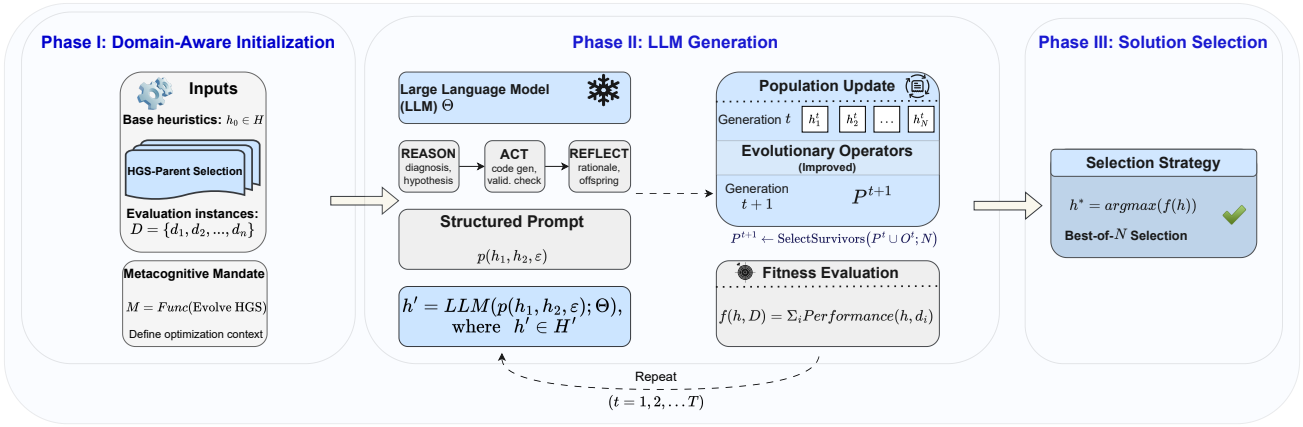

Figure 1: An overview of MEP, which outlines a process for evolving high-performing heuristics. The process continues until convergence criteria or a maximum number of generations is reached, ultimately returning the best-performing heuristic.

constraint reflects practical considerations in logistics, where vehicles must deliver goods before they have space to collect returns or recyclables.

- **VRP with Time Windows (VRPTW)**: Each customer i must be visited within a specified time window $[a_i, b_i]$, where a_i is the earliest service time and b_i is the latest service time. If a vehicle arrives before a_i, it must wait until the service can begin. Arrival after b_i renders the solution infeasible. This variant is particularly relevant in applications such as package delivery, where customers have preferred delivery times, or in service industries with appointment scheduling.

A formal mathematical definition of the base VRP is provided in Appendix A.

3.3 Hybrid Genetic Search

HGS is a powerful metaheuristic framework that combines the global search capabilities of a genetic algorithm with the fine-tuning power of a highly effective local search. The general workflow of HGS is as follows:

- *Initialization:* A diverse initial population is generated, often using problem-specific constructive heuristics.
- *Parent Selection:* Two parent solutions (P_1,P_2) are selected using a fitness-biased mechanism that balances solution quality (elitism) and genetic diversity, typically measured by solution distance metrics.
- *Crossover:* An offspring solution (C) is generated using problem specific recombination operators, such as Selective Route-EXchange (SREX) for VRPs, that effectively preserve meaningful structural characteristics of the parent solutions.
- *Education (Local Search):* The offspring undergoes intensive improvement via a problem-tailored local search procedure. For VRPs, this typically involves neighborhood moves like *relocate*, *swap*, and *2-opt* operators. This hybridization with local search is fundamental to HGS's performance.
- *Survivor Selection:* The educated offspring is considered for population inclusion. If population size limits are exceeded, removal prioritizes either low-quality solutions or those exhibiting high similarity to others, maintaining both quality and diversity through a distance-based diversity management mechanism.
- *Diversification:* When stagnation is detected, the search restarts through partial population reinitialization or injection of new diverse solutions, preventing premature convergence.

PyVRP is a state-of-the-art, open-source implementation of HGS specific to VRPs, written in Python with performance-critical components in C++. Its modular architecture enables isolated replacement of core components (e.g., parent selection, local search operators) with custom implementations, creating an ideal experimental platform for heuristic evolution via LLM-generated functions.

4 METACOGNITIVE EVOLUTIONARY PROGRAMMING

We propose Metacognitive Evolutionary Programming (MEP), an evolutionary framework inspired by Bai et al. [1] that emulates human metacognitive reasoning by guiding the LLM through a structured and strategic design process. As illustrated in Figure 1, MEP enables the LLM to evolve heuristics through a human-like reason–act–reflect cycle, augmented with domain knowledge and guided by a planning strategy that fosters strategic improvement.

The central thesis of MEP is that engaging the LLM in a structured, hypothesis-driven process makes code evolution more efficient and more likely to yield novel, high-performing heuristics. The framework is composed of two main phases: a one-time *Domain-Aware Initialization*, and an iterative *Reason–Act–Reflect cycle*.

4.1 Phase 1: Domain-Aware Initialization

Before the evolutionary search begins, MEP equips the LLM with comprehensive knowledge about the target component. This planning phase provides three critical knowledge types:

- *Common Pitfalls (K_p):* Known failure modes specific to the component being evolved.

- *Mitigation Strategies (K_s):* Proven techniques to address identified pitfalls.
- *Problem-Specific Traps (K_t):* Domain-specific challenges (e.g., in VRP solving).

This knowledge foundation serves as domain expertise that guides the LLM's reasoning throughout the evolutionary process, ensuring that generated heuristics are informed by established algorithmic principles.

4.2 Phase 2: Reason-Act-Reflect Cycle

Each generation of a new heuristic in MEP follows a structured three-step cognitive process, called the Reason-Act-Reflect cycle.

Step 1: REASON (Diagnosis and Hypothesis). At the start of each evolutionary timestep, two parent heuristics are selected from the current population. Before generating any code, the LLM is prompted to perform a structured analysis of the parents, encouraging deliberate reasoning over random mutation.

- *Diagnosis:* The LLM begins by analyzing the provided parent heuristics, i.e., $\texttt{code}_1$, $\texttt{code}_2$, along with their performance scores and any available feedback. It is explicitly prompted to diagnose their weaknesses, guided by the principles introduced in *Phase 1*.
- *Design Hypothesis:* Building on the identified weaknesses, the LLM must formulate a concise, one-sentence *Design Hypothesis* that proposes a specific improvement for the new heuristic ($\texttt{code}_3$). This step is crucial, as it ensures the LLM explicitly reasons about how to address the limitations of its predecessors, rather than simply combining elements of existing code.

Step 2: ACT (Code Implementation). Once the hypothesis is formulated, the LLM is tasked with implementing it. This step is governed by strict guardrails including a fixed function signature, approved libraries, and a required output format, to ensure the generated code is syntactically valid and fully compatible with the PyVRP evaluation sandbox.

Step 3: REFLECT (Rationale and Self-Critique). After generating the code, the LLM engages in a final metacognitive step: self-reflection. In this phase, it critically evaluates its prior reasoning, hypothesis formulation, and implementation choices. This self-assessment is recorded and carried forward to inform the generation of future offspring, enabling improvement over subsequent evolutionary cycles.

The Reason-Act-Reflect cycle enforces a principled generation process, ensuring that each new heuristic emerges from structured reasoning rather than arbitrary mutation. Figure 1 outlines the process for evolving high-performing heuristics using the proposed method. A high-level prompt structure is presented in Figure 2, breaking-down the LLM interaction into three distinct phase-aligned segments. *Planning* provides high-level strategic context, explicitly injecting domain knowledge to ground the LLM's initial reasoning in established principles. *Reasoning* directs the LLM to perform a structured diagnosis of parent heuristics' weaknesses and mandates the formulation of a concise, testable *Design Hypothesis* before any code generation, ensuring deliberate improvement intent. *Reflection* requires the LLM to self-critique its generated code and the preceding reasoning/hypothesis within the code documentation, embedding metacognitive assessment directly into the output. Each segment combines high-level instructions (e.g., "Diagnose the weaknesses...") with detailed technical specifications (e.g., fixed function signatures, approved libraries, required output format) to enforce both strategic coherence and syntactic correctness. The full prompt is provided in Appendix B.

Role: AI Optimization Researcher (Vehicle Routing Problem)

1) Planning Phase

- **Pitfalls (K_p):** List potential failures in VRP parent selection...
- **Strategies (K_s):** Propose countermeasures for each pitfall...
- **Hidden Traps (K_t):** Identify misleading problem instance features...

Overall Objective: Write a `novel` Python function `select_parents` for a Hybrid Genetic Search algorithm, guided by the planning insights.

Context: Background on parent selection in genetic algorithms for VRP, including standard approaches (e.g., elitist, tournament). ...

Common Existing Approaches: Standard parent selection methods in genetic algorithms include elitist selection (top-ranked individuals), tournament selection (best from random subset), fitness-proportionate selection, etc. ...

2) Reasoning Phase

- **Analyze Patterns:** Assess exploration/exploitation bias in provided selectors.
- **Assess Shortcomings:** Evaluate selectors against the K_p/K_t list.
- **Design Sketch:** Outline the proposed new selector, `sel3`, in prose.

Your Task in This Interaction:

1. Analyze `sel1` and `sel2`.
2. Design and implement `sel3 = select_parents(...)`.
3. **Implementation Requirements:**
   - **Inputs:** `population`, `rng`, `cost_evaluator`, `k=2`
   - **Output:** `tuple[Solution, Solution]`
   - **Behavior:** Balance feasibility, cost, and diversity.

Reference Python Classes:

```
class CostEvaluator: ...
class Solution: ...
class ProblemData: ...
# ... (other PyVRP class definitions)
```

3) Reflection Phase

1. How does the final code address the initial pitfalls (K_p)?
2. Did new challenges emerge during coding?
3. Propose two concrete revisions for further improvement.

Response Format: A single Python code block containing only the required function and imports.

Dynamic Inputs for a Given Task:

- **Selector 1:** `$code1`
- **Performance:** Cost: `$score1`, Feedback: `$feedback1`
- **Selector 2:** `$code2`
- **Performance:** Cost: `$score2`, Feedback: `$feedback2`

Figure 2: The prompt template used to guide the LLM. It consists of distinct instructions for planning, reasoning, and reflection, providing both high-level guidance and detailed technical specifications.

5 MEP CASE STUDY ON HGS

We apply MEP to evolve three critical components within the HGS algorithm. This includes `select_parents` for fitness-diversity balanced selection, `select_survivors` for population management, and `update_penalties` for constraint handling of infeasible solutions. These operators directly govern the exploration-exploitation trade-off of the genetic algorithm. In this context, exploitation refers

Table 1: Average cost comparison of the original PyVRP components and the best components evolved by MEP on 100 TSP instances. Lower cost values indicate better performance.

Component Evolved	Baseline Cost	Best Evolved Cost	Improvement
`select_parents`	7 942 166	**7764815**	**2.23 %**
`select_survivors`	7 942 166	**7886852**	**0.69 %**
`update_penalties`	7 942 166	**7932973**	**0.12 %**

to selecting high-quality (low-cost) parents to refine the best-known solutions, while exploration involves selecting diverse parents to introduce new genetic material and prevent premature convergence to local optima. A better heuristic must balance these two competing objectives. To this end, we use the **PyVRP** library as our experimental sandbox. Thanks to its modular design, we can isolate and replace core HGS components (such as the parent selection operator) with LLM-generated functions. This approach creates PyVRP$^+$, a collection of better-performing heuristics.

Evolutionary Process: It is outlined as follows:

- **Base Population:** The base population consists of default implementations of selected modules from PyVRP.
- **Evaluation:** Each heuristic in the population is evaluated by running HGS on a set of VRP instances, with its fitness score defined as the negative average cost of the solution achieved.
- **Generation:** In each generation, two parent heuristics and their performance scores are selected. These, together with domain knowledge, are provided to the MEP prompt to produce a new offspring heuristic.
- **Population Update:** The new heuristic is evaluated and considered for inclusion in the population.
- **Termination:** The process runs for a fixed number of generations (N), and the best heuristic discovered is reported.

The evolutionary process runs for 10 generations. In each generation, 10 offspring heuristics are produced, and the top 5 performers, i.e., selected from both the new offspring and the existing population, are retained for the next generation. To ensure the robustness of our findings, the entire evolutionary process for each component was conducted five times using different random seeds, and the best-performing heuristic from these runs is reported in our results. The LLM interactions were performed using the GPT-4.1 model. We set the temperature parameter to 1.0 to encourage creative and diverse heuristic designs, while all other parameters, such as top-p and max_tokens, were left at their default values.

Benchmark Suite: We evaluate our approach on two datasets. First, we use the publicly available TSP100 dataset [30], a well-established benchmark in combinatorial optimization, to assess LLM-generated heuristics during the evolution process. Second, we evaluate the discovered HGS variants on a comprehensive benchmark suite spanning six VRP variants. Each variant, sourced from the PyVRP library[1], includes a dedicated folder containing 60 instances, ensuring statistically robust evaluation. The benchmark spans a range of problem complexities, i.e., from the standard Capacitated VRP (CVRP) to more challenging formulations like the Multi-Depot VRP with Time Windows (MDVRPTW). The diversity of these variants, each with distinct constraint structures and objectives, provides a comprehensive testing ground for evaluating the robustness and generalizability of evolved heuristics.

6 EXPERIMENTAL RESULT AND ANALYSIS

Our primary goal is to demonstrate that MEP can discover better heuristic components for a SOTA VRP solver. Before presenting the results, it is crucial to distinguish between two different time costs: the one-time design cost of the evolutionary process and the recurring execution cost of the final heuristic. The performance tables in our analysis report the execution cost, which is the wall-clock time required for the final, evolved heuristic to solve a given VRP instance. This metric allows for a direct comparison against the baseline solver's runtime. In contrast, the design cost is the upfront, one-time computational investment required to discover the heuristic. This involves the entire MEP process, including all LLM API calls and the evaluation of candidate solutions over multiple generations. We consider this design cost analogous to the manual effort a human researcher would invest over weeks or months to develop a novel algorithm. For this work, the total design cost for discovering each final component was approximately 3–4 hours on a single NVIDIA H100 GPU. This involved approximately 100 LLM API calls per component at a total cost of $15–20 USD, substantially lower than the weeks of manual effort typically required. We posit that this one-time investment is a practical trade-off for producing novel heuristics that significantly improve solution quality and reduce subsequent execution costs. We use GPT-4.1 in the evolutionary search, as mentioned earlier, chosen for its strong performance and cost-effectiveness. The following standard metrics are used to evaluate solution quality:

- **Cost/Obj.:** The total cost of a solution, typically representing the sum of travel distances or travel times across all routes. Lower values indicate better solutions.
- **Improvement:** The relative percentage by which the method's objective value is better than the best-known solution (BKS). It is computed as:

$$\text{Improvement} = \frac{\text{Obj}_{\text{BKS}} - \text{Obj}_{\text{method}}}{\text{Obj}_{\text{BKS}}} \times 100\% \quad (1)$$

A positive value indicates that the method outperforms the BKS, while a negative value means it performs worse.

[1] https://github.com/PyVRP/Instances/

Table 2: Performance comparison of the baseline PyVRP solver versus the PyVRP+ equipped with the MEP-evolved operators across various VRP variants (average values are presented). Lower cost and shorter times indicate better performance.

Variant	Baseline Cost	MEP-Evolved Cost	Improvement	Baseline Time (s)	MEP-Evolved Time (s)
CVRP	64 450.03	**64236.30**	0.33 %	9.55	**9.11**
GVRP	7 686 993.64	**7532007.09**	2.01 %	**0.87**	1.89
MDVRPTW	8955.11	**8825.46**	1.45 %	12.43	**9.74**
PCVRPTW	22 975.85	**22406.42**	2.48 %	7.92	**4.19**
VRPB	83 696.21	**83146.06**	0.66 %	14.07	**13.85**
VRPTW	33 424.48	**32521.75**	2.70 %	17.03	**14.66**

6.1 Hardware Settings

All experiments are conducted on a high-performance computing cluster with the following specifications:

- **GPU:** NVIDIA H100 GPU with dedicated allocation
- **CPU:** 32-core processor allocation per job
- **Memory:** 120 GB RAM per compute node
- **Cluster Configuration:** Single-node allocation with GPU acceleration

6.2 Evolving Effective HGS Components

Table 1 summarizes the average cost of the best-evolved heuristic for each target component compared to the PyVRP baseline on the TSP100 benchmark. In all cases, MEP successfully discovered an operator with a better cost. The most significant improvements were observed in the `select_parents` and `select_survivors` components, which saw a 2.23% and 0.69% improvement, respectively. This highlights MEP's ability to innovate on complex, high-impact components of the metaheuristic. The cost-generation curve illustrating the iterative improvement of the evolutionary process is shown in Figure 3.

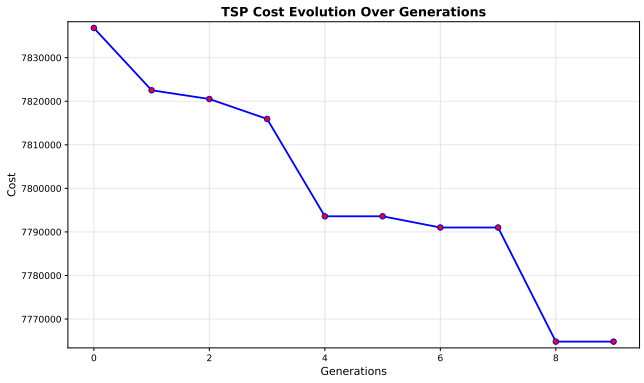

Figure 3: Evolution of average cost values over 10 generations for TSP optimization using MEP for parent selection, with averages computed across 100 instances.

A key outcome of the MEP process is the emergence of sophisticated and novel heuristics that go beyond simple variations of existing strategies. For example, the top-performing `select_parents` operator, described by the LLM as a "Hybrid Adaptive Stratified Diversity-Pressure Selection", divides the population into cost-based strata and applies tailored diversity pressures to each subgroup. The complete evolved code is provided in Appendix C. These results highlight that MEP's hypothesis-driven framework enables the discovery of complex, well-motivated algorithmic innovations, far beyond simple parameter tuning or superficial modifications.

To evaluate the synergistic potential of these evolved heuristics, we created an integrated HGS solver that combines the best-performing LLM-generated modules for parent selection, survivor selection, and penalty adjustment. This integrated solver represents a more holistic test of MEP's capabilities, assessing whether components evolved in isolation can cooperate effectively. As we will demonstrate in the following section, this integrated approach not only matches but often exceeds the performance of individually evolved modules, achieving consistent cost reductions and significant runtime improvements on complex VRP variants. This finding validates that MEP can produce a suite of compatible, high-performing heuristics that collectively advance the state-of-the-art.

6.3 Integrated Solver Performance and Generalization

To assess the robustness and overall performance of our approach, we evaluated the fully integrated HGS solver, i.e., equipped with the best evolved heuristics for parent selection, survivor selection, and penalty updates against the PyVRP baseline on a diverse set of VRP variants. The results, shown in Table 2, demonstrate a clear and consistent advantage for the MEP-evolved solver.

The integrated solver achieves notable cost reductions across all variants, with particularly strong improvements on complex instances like VRPTW (2.70%), PCVRPTW (2.48% improvement) and GVRP (2.01% improvement). Furthermore, the evolved heuristics lead to significant computational efficiencies, reducing runtime by over 45% on PCVRPTW and 10% on MDVRPTW. While some variants show a modest increase in runtime, the superior solution quality underscores the effectiveness of the discovered strategies. This strong generalization performance confirms that MEP is capable of producing a synergistic set of heuristics that are not over-fitted to a single problem type and can enhance a state-of-the-art solver across a wide range of operational scenarios.

6.4 Generalization Across Evolved Components

We also evaluated the generalization capabilities of the other best-evolved. Two of these experimental results are presented in Table 4, which also demonstrate positive, albeit more modest, improvements

Table 3: Ablation study results for the `select_parents` operator, showing percentage improvement over the baseline. Average cost values are presented in the table.

Variant	Baseline Cost	Full MEP Impr.	MEP-noInit Impr.	Reactive Evol. Impr.
CVRP	64 450.03	**0.33 %**	0.28 %	−0.20 %
GVRP	7 686 993.64	**1.82 %**	0.98 %	1.30 %
MDVRPTW	8955.11	**1.49 %**	0.90 %	−2.05 %
PCVRPTW	22 975.85	**2.43 %**	2.36 %	1.46 %
VRPB	83 696.21	**0.57 %**	0.53 %	−0.39 %
VRPTW	33 424.48	**2.25 %**	2.24 %	0.10 %

Table 4: Generalization performance of individual versus combined MEP-evolved operators. The table shows the percentage improvement over the HGS baseline, with the final column (PyVRP$^+$) representing a solver that integrates all individually evolved components.

Variant	select_parents	select_survivors	update_penalties	PyVRP$^+$
CVRP	**0.33 %**	0.05 %	0.01 %	**0.33 %**
GVRP	1.82 %	0.08 %	0.13 %	**2.01 %**
MDVRPTW	**1.49 %**	0.00 %	0.16 %	1.45 %
PCVRPTW	2.43 %	0.15 %	0.06 %	**2.48 %**
VRPB	0.57 %	0.04 %	0.05 %	**0.66 %**
VRPTW	2.25 %	0.29 %	0.35 %	**2.70 %**

over the baseline across various VRP instances. For example, the evolved ‘select_survivors‘ operator improves VRPTW performance by 0.29%, while the evolved ‘update_penalties‘ operator improves it by 0.35%. This indicates that the MEP framework is capable of discovering generally beneficial heuristics for multiple components of the HGS algorithm, even if the primary gains are concentrated in the highest-impact operators, such as parent selection.

Notably, none of the evolved HGS components underperformed their baseline counterparts on any of the six VRP variants, despite not being explicitly evaluated on these during the evolution process. This highlights the robustness and generalizability of the discovered algorithmic strategies.

6.5 Ablation Studies

To assess the individual contributions of key components in our framework, we performed a series of ablation studies focused on the select_parents module, chosen for its strong generalization across tasks. These experiments aim to isolate the effects of the Domain-Aware Initialization and the structured Reason–Act–Reflect cycle. Our ablation compares full MEP framework against two variants:

- MEP-noInit: This version removes the Domain-Aware Initialization from the prompt. The LLM still performs the Reason-Act-Reflect cycle, but its reasoning is purely local, based only on the two parent heuristics provided in each generation.
- Reactive Evolution: This variant removes the structured reasoning components. The prompt is simplified, providing the parent heuristics and their scores and asking the LLM to generate an improved version. This setup mimics prior reactive evolution methods such as EoH [13] and ReEvo [32].

The results in Table 3 reveal a clear performance hierarchy. The full MEP framework consistently achieves the best results. The MEP-noInit variant performs reasonably well but is worse than the full MEP, highlighting the benefit of the strategic guidance provided by the Domain-Aware Initialization. For example, on GVRP, the improvement drops from 1.82% with the full MEP to 0.98% without the Initialization. The Reactive Evolution variant performs the worst, often yielding only marginal gains or even performance degradation (e.g., -0.20% on CVRP), underscoring the importance of the structured reason-act-reflect cycle.

7 CONCLUSION

In this work, we introduced Metacognitive Evolutionary Programming (MEP), a new paradigm for automated algorithm design that enhances LLM-driven evolution with strategic planning and structured reasoning for the Vehicle Routing Problem (VRP). By grounding the LLM in domain knowledge and guiding it through a structured Reason–Act–Reflect cycle, we elevate it from a basic code mutator to a strategic agent capable of deliberate heuristic discovery. Our application of MEP to evolve core components of the state-of-the-art HGS solver demonstrates a promising path toward discovering heuristics that are not only high-performing but also novel and strategically sound. Looking ahead, this work opens up several avenues for future research. Our work demonstrates that MEP can successfully enhance individual components of a state-of-the-art solver. More importantly, we have shown that these independently evolved heuristics can be successfully integrated, creating a holistically improved solver that generalizes across multiple challenging VRP variants. Future work could explore the joint, simultaneous evolution of all components to uncover even deeper synergistic interactions, though this remains a computationally intensive challenge. While this presents significant challenges with the current LLM context length and computational costs, we believe it is crucial for uncovering synergistic interactions between components. Our work demonstrates that a manageable one-time design cost can yield heuristics with lasting performance benefits, providing a strong incentive to tackle these future challenges. Furthermore, to rigorously test the generalizability of our discovered heuristics, we plan to evaluate their performance across the full set of 48 VRP variants covered in the original PyVRP. Overall, this work lays a foundation for more autonomous and intelligent algorithmic systems capable of scientific discovery, transcending the constraints imposed by manually designed algorithms. The code is available at https://github.com/ra-MANUJ-an/pyvrp-code.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under the AI Singapore Programme (AISG Award No: AISG3-RPGV-2025-017), and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

[1] Tian Bai, Yongwang Cao, Yan Ge, and Haitao Yu. 2025. MP: Endowing Large Language Models with Lateral Thinking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 23460–23468.

[2] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* 290, 2 (2021), 405–421.

[3] Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Qingfu Zhang, Joungho Kim, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Kevin Tierney, Lin Xie, and Jinkyoo Park. 2025. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. In *SIGKDD Conference on Knowledge Discovery and Data Mining*.

[4] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels A Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. 2025. RouteFinder: Towards Foundation Models for Vehicle Routing Problems. *Transactions on Machine Learning Research* 2025 (2025).

[5] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. 2023. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research* 24, 130 (2023), 1–61.

[6] Angelica Chen, David Dohan, and David So. 2023. EvoPrompting: Language Models for Code-Level Neural Architecture Search. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=ifbF4WdT8f

[7] Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. 2025. Large Language Models for Combinatorial Optimization: A Systematic Review. *arXiv preprint arXiv:2507.03637* (2025).

[8] 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*, Vol. 39. 26931–26938.

[9] Alexander David Goldie, Zilin Wang, Jakob Nicolaus Foerster, and Shimon Whiteson. 2025. How Should We Meta-Learn Reinforcement Learning Algorithms?. In *Reinforcement Learning Conference*. https://openreview.net/forum?id=jKzQ6af2DU

[10] Jin Huang, Qihao Liu, Xinyu Li, Liang Gao, and Yue Teng. 2026. Automatic programming via large language models with population self-evolution for dynamic fuzzy job shop scheduling problem. *IEEE Transactions on Fuzzy Systems* (2026).

[11] Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. 2025. Large language models as end-to-end combinatorial optimization solvers. In *Advances in Neural Information Processing Systems*.

[12] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. 2023. Algorithm evolution using large language model. *arXiv preprint arXiv:2311.15249* (2023).

[13] Fei Liu, Tong Xialiang, 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. In *International Conference on Machine Learning*.

[14] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery. *arXiv preprint arXiv:2408.06292* (2024). arXiv:2408.06292 [cs.AI] https://arxiv.org/abs/2408.06292

[15] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shikha Prabhumoye, Yiming Yang, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*.

[16] Kalyan Varma Nadimpalli, Shashank Reddy Chirra, Pradeep Varakantham, and Stefan Bauer. 2025. Evolving RL: Discovering New Activation Functions using LLMs. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*. https://openreview.net/forum?id=H2x9juCuJg

[17] Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, et al. [n.d.]. MLGym: A New Framework and Benchmark for Advancing AI Research Agents. In *Second Conference on Language Modeling*.

[18] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131* (2025).

[19] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475.

[20] Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, and Jie Zhang. 2025. Generalizable Heuristic Generation Through Large Language Models with Meta-Optimization. *arXiv preprint arXiv:2505.20881* (2025).

[21] Yiwen Sun, Xianyin Zhang, Shiyu Huang, Shaowei Cai, BingZhen Zhang, and Ke Wei. 2024. AutoSAT: Automatically optimize sat solvers via large language models. *arXiv preprint arXiv:2402.10705* (2024).

[22] Anja Šurina, Amin Mansouri, Lars CPM Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. [n.d.]. Algorithm Discovery With LLMs: Evolutionary Search Meets Reinforcement Learning. In *Second Conference on Language Modeling*.

[23] Nguyen Thach, Aida Riahifar, Nathan Huynh, and Hau Chan. 2025. RedAHD: Reduction-Based End-to-End Automatic Heuristic Design with Large Language Models. *arXiv preprint arXiv:2505.20242* (2025).

[24] Thibaut Vidal. 2022. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research* 140 (2022), 105643.

[25] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*.

[26] Niels A Wouda, Leon Lan, and Wouter Kool. 2024. PyVRP: A high-performance VRP solver package. *INFORMS Journal on Computing* 36, 4 (2024), 943–955.

[27] Xuan Wu, Di Wang, Chunguo Wu, Lijie Wen, Chunyan Miao, Yubin Xiao, and You Zhou. 2025. Efficient Heuristics Generation for Solving Combinatorial Optimization Problems Using Large Language Models. In *SIGKDD Conference on Knowledge Discovery and Data Mining*.

[28] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025. The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search. *arXiv preprint arXiv:2504.08066* (2025). arXiv:2504.08066 [cs.AI] https://arxiv.org/abs/2504.08066

[29] Xianliang Yang, Lei Song, Yapu Zhang, and Jiang Bian. 2025. HeurAgenix: A Multi-Agent LLM-Based Paradigm for Adaptive Heuristic Evolution and Selection in Combinatorial Optimization. https://openreview.net/forum?id=xxSK3ZNAhh

[30] Shunyu Yao, Xi Lin, Jiashu Wang, Qingfu Zhang, and Zhenkun Wang. 2024. Rethinking supervised learning based neural combinatorial optimization for routing problem. *ACM Transactions on Evolutionary Learning* (2024).

[31] Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. 2025. Multi-objective evolution of heuristic using large language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 27144–27152.

[32] 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 *Advances in Neural Information Processing Systems*.

[33] Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. 2025. Large Language Model-driven Large Neighborhood Search for Large-Scale MILP Problems. In *International Conference on Machine Learning*.

[34] Ni Zhang, Zhiguang Cao, Jianan Zhou, Cong Zhang, and Yew-Soon Ong. 2025. An Agentic Framework with LLMs for Solving Complex Vehicle Routing Problems. In *International Conference on Learning Representations*.

[35] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. In *International Conference on Machine Learning*.

[36] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. 2024. MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts. In *International Conference on Machine Learning*.

[37] Rongjie Zhu, Cong Zhang, and Zhiguang Cao. 2025. Refining Hybrid Genetic Search for CVRP via Reinforcement Learning-Finetuned LLM. In *International Conference on Learning Representations*.