Bridging Synthetic and Real Routing Problems via LLM-Guided Instance Generation and Progressive Adaptation

Jianghan Zhu[1*], Yaoxin Wu[2], Zhuoyi Lin[3†], Zhengyuan Zhang[4], Haiyan Yin[5], Zhiguang Cao[1], Senthilnath Jayavelu[3,7], Xiaoli Li[4,6]

[1]Singapore Management University, Singapore
[2]Eindhoven University of Technology, Netherlands
[3]Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore
[4]Nanyang Technological University, Singapore
[5]Centre for Frontier AI Research (CFAR), Agency for Science, Technology and Research (A*STAR), Singapore
[6]Singapore University of Technology and Design, Singapore
[7]National University of Singapore, Singapore
{zhuj0044, zhengyua002}@e.ntu.edu.sg, y.wu2@tue.nl, {Lin_Zhuoyi, Yin_Haiyan, J_Senthilnath}@a-star.edu.sg, zgcao@smu.edu.sg, xiaoli_li@sutd.edu.sg

Abstract

Recent advances in Neural Combinatorial Optimization (NCO) methods have significantly improved the capability of neural solvers to handle synthetic routing instances. Nonetheless, existing neural solvers typically struggle to generalize effectively from synthetic, uniformly-distributed training data to real-world VRP scenarios, including widely recognized benchmark instances from TSPLib and CVRPLib. To bridge this generalization gap, we present Evolutionary Realistic Instance Synthesis (**EvoReal**), which leverages an evolutionary module guided by large language models (LLMs) to generate synthetic instances characterized by diverse and realistic structural patterns. Specifically, the evolutionary module produces synthetic instances whose structural attributes statistically mimics those observed in authentic real-world instances. Subsequently, pre-trained NCO models are progressively refined, firstly aligning them with these structurally enriched synthetic distributions and then further adapting them through direct fine-tuning on actual benchmark instances. Extensive experimental evaluations demonstrate that EvoReal markedly improves the generalization capabilities of state-of-the-art neural solvers, yielding a notable reduced performance gap compared to the optimal solutions on the TSPLib (1.05%) and CVRPLib (2.71%) benchmarks across a broad spectrum of problem scales.

Code — https://github.com/HenryZhu1029/EvoReal
Extended version — https://arxiv.org/abs/2511.10233

1 Introduction

Being a longstanding NP-hard challenge, Vehicle Routing Problems (VRPs) represent a classic family of combinatorial optimization problems (COPs) (Bengio, Lodi, and Prouvost 2021; Cappart et al. 2023) widely encountered in the field of logistics (Cattaruzza et al. 2017) and public transportation (Konstantakopoulos, Gayialis, and Kechagias 2022). In recent years, Neural Combinatorial Optimization (NCO) methods have demonstrated remarkable success in tackling various VRPs through deep reinforcement learning and attention-based models, achieving state-of-the-art performance on synthetic VRP instances (Bello et al. 2017; Kool, van Hoof, and Welling 2019; Li, Yan, and Wu 2021; Drakulic et al. 2023; Ma, Cao, and Chee 2023). Despite this progress, existing NCO models frequently exhibit limited generalization when transitioning from synthetic, uniformly distributed training data to real-world problem instances, such as those found in benchmarks like TSPLib (Reinelt 1991) and CVRPLib (Uchoa et al. 2017). This distributional shift significantly restricts their practical applicability (Joshi et al. 2006; Bi et al. 2022; Gao et al. 2024).

On the other hand, LLM solvers exhibit strong generalization capabilities when confronted with new COPs and distributions. Recent works leverage LLMs to dynamically generate heuristics or evolve hyper-heuristics (Duflo et al. 2019; Drake et al. 2020), thus facilitating efficient Evolutionary Search (ES) of heuristic spaces without human bias (Yang et al. 2024). Noteworthy endeavors include FunSearch (Romera-Paredes et al. 2024), EoH (Liu et al. 2024a), and ReEvo (Ye et al. 2024). However, existing LLM-based approaches often encounter challenges when handling tasks that require extensive contextual information, typically demonstrating reduced coherence and accuracy in the processing of extended textual descriptions (Yang et al. 2024; Xu et al. 2025). Consequently, these approaches exhibit significant limitations when addressing medium- or large-size instances (e.g., instances involving more than 50 nodes), thus compromising their ability to effectively generalize to real-world VRP instances.

Motivated by these observations, we propose EvoReal, a novel data-centric framework explicitly designed to enhance the generalization capabilities of NCO models for real-

[*]Work done during internship at A*STAR.
[†]Zhuoyi Lin is the corresponding author.

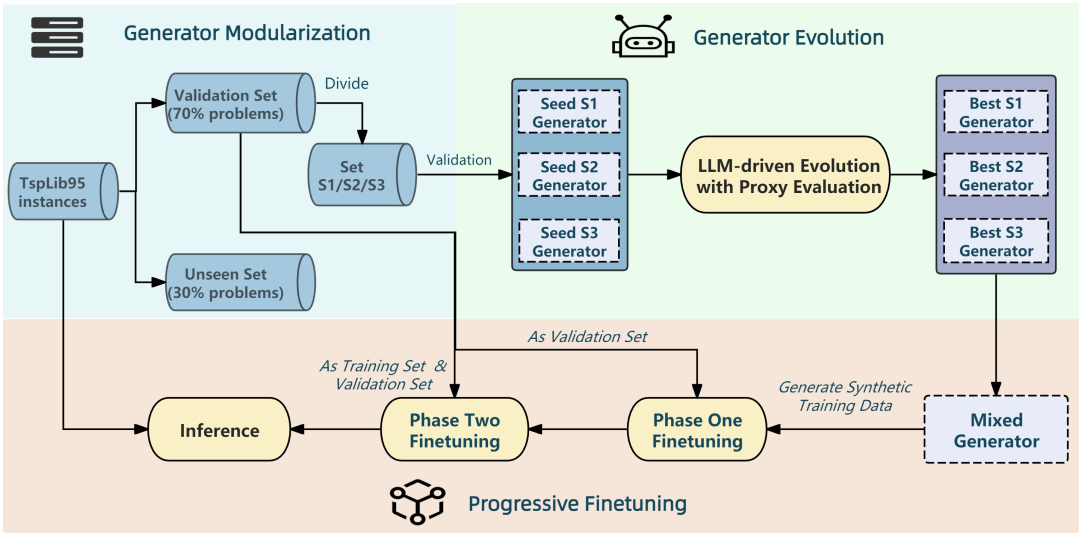

Figure 1: Overall workflow of EvoReal including LLM-guided generator evolution and progressive fine-tuning. **Top Left**: Validation set and unseen test set are split, with validation problems grouped by distribution category for structural-specific generator design. **Top Right**: LLM-driven module evolves generators which are evaluated on specific validation sets. **Bottom**: Pre-trained models are progressively fine-tuned on data from the evolved generators and the validation set's real instances.

world VRPs. Unlike prior approaches that evolve heuristics or solvers for direct solution construction, we innovatively leverage LLMs to evolve data generators that facilitate model adaptation. At the core of EvoReal is an LLM-driven evolutionary module, which generates synthetic VRP instances structurally aligned with real-world distributions. This module systematically *evolves simplistic synthetic scenarios into diverse and complex distributions*, effectively capturing the intricate characteristics of real-world datasets. Subsequently, we introduce a progressive fine-tuning strategy which incrementally adapts pre-trained neural solvers by transitioning through increasingly complex synthetic instances toward actual real-world VRP scenarios. Consequently, EvoReal *facilitates smoother model adaptation by progressively refining solver parameters and representations*, effectively bridging the generalization gap between synthetic data and real-world benchmark instances. We summarize our contributions as follows. (1) We propose EvoReal, an LLM-guided evolutionary framework to synthesize structurally realistic VRP instances, bridging the distributional gap between synthetic and real-world routing problems. (2) We introduce a progressive fine-tuning strategy, which first adapts neural combinatorial solvers to diverse LLM-evolved distributions and then further refines them on real benchmark data, enabling effective domain adaptation without architectural changes. (3) Extensive experiments on TSPLib and CVRPLib benchmarks demonstrate that EvoReal significantly improves the generalization of SOTA neural solvers across a wide range of problem sizes, achieving new state-of-the-art results and notably reducing the performance gap between small and large instances. Notably, our results demonstrate that this generator-based adaptation consistently outperforms direct fine-tuning on real data alone.

2 Related Works

Constructive Neural VRP Solvers. Those models learn policies to construct solutions in a step-by-step or one-shot manner: Pointer Network (Ptr-Net) proposed by (Vinyals, Fortunato, and Jaitly 2015) employed an encoder-decoder network that generates solutions to VRPs sequentially. Follow-up works utilize reinforcement learning (RL) to improve Ptr-Net's gradient update to generate better approximate solutions on TSP (Bello et al. 2017) and CVRP (Nazari et al. 2018). More recently, NCO solvers have shifted toward attention-based architectures, starting with the attention model (AM) (Kool, van Hoof, and Welling 2019), which is versatile on a wide range of COPs. Policy optimization with multiple optima (POMO) (Kwon et al. 2020), has further promoted the capability of the attention-based model on TSP and CVRP. Other constructive frameworks developed from AM and POMO illustrate their broader scalability (Kwon et al. 2021; Choo et al. 2022; Kim, Park, and Park 2022; Chen et al. 2023; Chalumeau et al. 2023; Hottung, Mahajan, and Tierney 2024; Lin et al. 2024).

Recent studies have investigated the severe generalization ability decrement in unseen problem sizes or distributions (Joshi et al. 2006; Liu et al. 2023). In the line of size generalization, there are many attempts to generalize solvers from small instances to larger ones (Lisicki, Afkanpour, and Taylor 2020; Kim et al. 2022; Bdeir, Falkner, and Schmidt-Thieme 2022; Hou et al. 2023; Son et al. 2023). Unlike approaches focused on scaling, considerable efforts have been devoted to addressing cross-distribution generalization, including works that augment training instances on multi-distribution and multitasks (Goh et al. 2025), and works which synergetically train a backbone model with multi-distribution instances towards a generalizable solver on out-of-distribution tasks (Zhou et al. 2023, 2024).

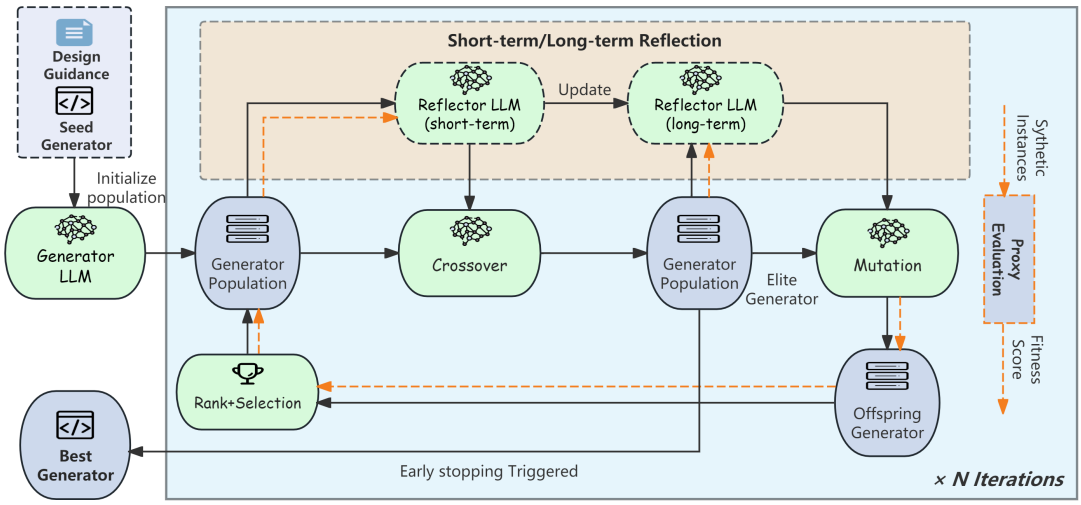

Figure 2: LLM-driven evolution component in Fig.1. The pipeline within the blue rectangle block is repeated for N iterations. Dotted arrows represent the proxy-evaluation of each generator; black arrows indicate the flow of generators. For each pair of parents, the reflector LLM performs short-term reflection based on their relative performance, and this insight is used to guide crossover for designing new offspring. Accumulated short-term reflections are further distilled into long-term ones, which guide mutation to improve the current best generator. After mutation, populations are ranked and selected to maintain a fixed size.

Learning to Optimize with LLMs. Existing approaches to improving generalization in combinatorial optimization (CO) take advantage of the linguistic comprehension and domain-specific competence of LLMs via prompt engineering to tackle COPs. These methods can be mainly divided into two mainstreams: 1) *LLMs are solution generators*. Early explorations in this line of research involve prompting LLMs to solve graph-based COPs (Wang et al. 2023), iteratively refining current heuristics (Iklassov et al. 2024), progressively improving solutions by targeting better objectives (Yang et al. 2024; Liu et al. 2024b; Elhenawy et al. 2024b), and instilling LLMs with visual modalities of problems and solutions (Elhenawy et al. 2024a; Huang et al. 2024). Recently, training LLM for end-to-end CO has been attempted, moving beyond prompting paradigms (Jiang et al. 2025a). 2) *LLMs are heuristic designers*. Explainable, task-oriented heuristics in the form of codes are iteratively improved throughout automatic heuristic generation and evaluation of Evolution Algorithms (EA), the productivity of which outstrips that of human experts (Romera-Paredes et al. 2024; Liu et al. 2024a; Ye et al. 2024; Tran et al. 2025; Jiang et al. 2025b). Among them, ReEvo (Ye et al. 2024) achieves strong adaptability in many COPs under both white-box and black-box prompt settings. In this paper, we focus mainly on the utilization of LLMs as code designers. Distinct from prior constructive neural solvers and LLM-as-optimizer approaches, our method leverages LLMs to evolve data generators, enabling a novel, generator-based adaptation pipeline that outperforms direct fine-tuning on real data.

3 Methodology

In this section, we propose a novel LLM-guided evolution framework (EvoReal) that evolves VRP data generators to generate VRPLib-style distributions and a progressive fine-tuning strategy that gradually shifts the generalization ability of the pre-trained neural model on synthetic uniform instances towards VRPLib. The proposed LLM-guided evolution approach is primarily comprised of two components: LLM-driven evolutionary search and proxy evaluation. Specifically, the **LLM-driven evolution** aims at gradually developing structurally specific generators, while in the **proxy evaluation**, performance of each LLM-designed generator is accessed throughout training and monitoring the progressive validation results. In the following part, by taking EvoReal for TSP as an example, we elaborate the generator evolution and progressive fine-tuning framework.

Generator Modularization. Fig.1 shows the entire pipeline of our framework. For TSP, the evolution of data generators is driven by various carefully-written prompt strategies crafted for three types of generator, which generate different distributions of TSP. This ensures the coverage of generated TSP distributions that emulate TSPLib-like distributions. We combine the three evolved generators to produce mixed-distribution TSP data for fine-tuning. To ensure a wide range of problem sizes and distributions, we select 70 TSPLib problems of size less than 5000, and then carefully pick 48 problems ($\approx 70\%$) for validation while leaving the remaining 22 problems as a held-out unseen set. Among the 48 validation problems, we further categorize them into three types (i.e., S1, S2 and S3), each serving as a specific validation set for one type of generator. Detailed categorization methods are provided in the extended version. For CVRP, we randomly select 70% problems from CVRPLib (SetX) for validation and direct fine-tuning, and the remaining 30% are left as the unseen set. The specifics of the generation of CVRP are discussed in the extended version.

Generator Evolution

We prompt the LLMs to transform latent generator configurations into realistic routing distributions. Formally, let $h_\phi : \mathcal{Z} \to \mathcal{X}$ represent a data generator parameterized by an LLM model ϕ, mapping a latent variable $z \sim \mathcal{Z}$ (e.g. seed generator) into a synthetic routing instance $x = h_\phi(z)$. Throughout the evolution, we iteratively minimize the divergence between the synthetic distribution of $h_\phi(\mathcal{Z})$ and the target real-world distribution $\mathcal{D}_{\text{real}}$:

$$\min_{\phi} \mathcal{L}_{\text{evolve}}(\phi) = D\left(\mathcal{D}_{\text{real}},\, \mathcal{D}\left(h_\phi(\mathcal{Z})\right)\right) \tag{1}$$

where $D(\cdot\|\cdot)$ is a divergence measure (e.g., KL divergence, average gap). This dual-level optimization framework explicitly captures how heuristic generation indirectly optimizes the underlying combinatorial optimization problem by aligning evolved heuristics with real-world structures.

Representation of Generators. Each generator h is described by three components: 1) a function description defines the format of input and describes valid output, 2) a code implementing generation, and 3) a fitness score. The fitness score, defined as a function f of h (i.e. $f : h \to \mathbb{R}$), serves as the performance indicator of h on the validation set. An example of an evolved generator that outputs TSP problems is given in Fig.7 in section E of the extended version.

Proxy Evaluation. Following recent work (Ramji et al. 2024), we use an approach that leverages external proxy metrics such as gaps or objective values as the divergence measure $\mathcal{D}$ in Eq.1 to evaluate the quality of LLM-generated outputs, in lieu of direct human or reference-based evaluation. During the LLM-driven evolution, the proxy evaluation is instantiated by measuring the average gap on the validation set, which serves as the proxy metric. The gap is the gap between the objective value found and the best-known optimum. Each generator only fine-tunes the model for a small number of epochs (e.g. 10 epochs for LEHD), and the best average gap on the validation set of each generator during training is taken as the fitness score. Generators with lower fitness scores are retained according to a fixed population size, guiding generator selection without full fine-tuning. This low-fidelity evaluation strategy substantially accelerates the evolution search while presenting reliable indications of the proximity of the generated distributions to the real ones, as is the standard in the neural architecture search and hyperparameter optimization literature.

Population Initialization. Our evolutionary search framework begins with population initialization, during which a population of N generators $h_1, h_2, ..., h_N$ is initialized by incorporating initialization prompts. These prompts instill LLMs with the prior knowledge about the distributions that they are encouraged to explore and reproduce. Those prompts are composed of a trivial blueprint of the seed generator and a design guidance, each as a hint for generating possible structures of satisfactory distributions. Ablation studies in Section 5 show that external knowledge plays a significant role in improving the performance of generators.

Evolution Process. In order to balance the effectiveness of evolutionary search and the efficiency of fitness evaluation, we incorporate targeted modifications into ReEvo (Ye et al. 2024) framework. Ours introduces a ranking-based selection mechanism that increases the survival rate of elite generators, thereby accelerating the search process and reducing token costs. In addition, by postponing the selection stage until after mutation, we balance exploration and exploitation, allowing more newly initialized generators to participate in crossover and mutation before selection. Table 5 in Section 5 shows the elevated performance of the evolution framework equipped with these modifications. We perform the workflow in Fig.2 to search for the best generator by repeating the evolution part for N iterations. The whole evolution procedure can be summarized as the following steps:

- Step 1. Choose a specific type of generator to evolve. Initialize the code population by requesting LLMs to devise N generators that align with the initialization prompts.
- Step 2. Expand the generator population by four prompt strategies (i.e., crossover, mutation, short- & long-term reflection) as exploration and modification operators.
- Step 3. Each new generator is then evaluated on their structurally-specific validation set. Then it is added to the population if both the code and the fitness score are valid.
- Step 4. Select k offspring generators based on the rank of each generator individual. Ranks are calculated from fitness scores and transformed into the probability of being selected. The lower the fitness score, the more likely the generator individual will be selected.
- Step 5. Repeat Steps 2 to 4.
- Step 6. Stop the iteration once the number of the generators evaluated reaches pre-set maximum or when no better generator can be found for consecutive m iterations.

Once the evolutions are completed for all types of generator, we save the best generators for subsequent progressive fine-tuning. Detailed explanations of prompt strategies and evolution mechanisms are listed in the extended version.

Progressive Fine-tuning

After generator evolution, we progressively fine-tune neural models to real-world distributions. Ablation studies in Section 5 show that directly fine-tuning pre-trained neural solvers with TSPLib leads to limited generalization capability improvement compared to our progressive fine-tuning strategies. Through comprehensive experimental results, we demonstrate that our progressive fine-tuning framework effectively bridges the generalization gap between vanilla synthetic distributions like uniform and heterogeneous real-world instances without modifying model architectures. We divide our progressive strategies into two phases.

Phase One. In the first phase of progressive fine-tuning, the best-performing generator of all types evolved collaboratively generate large-scale synthetic data of the same problem size. By training pre-trained models to the disparate node patterns in these diverse, TSPLib-style instances, neural solvers are better equipped to handle challenging patterns in real-world scenarios through preliminary exposure to similar structures. Compared to the proxy evaluation conducted during LLM-driven generator evolution, phase one involves

Method	[0,200)		[200,500)		[500,1000)		[1000,5000)		Overall		Time
	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	
#Problems	27		15		6		22		70		
Concorde	30558.67	0.00%	42279.13	0.00%	29658.17	0.00%	158431.73	0.06%	73166.50	0.02%	8h 59min
LKH-3	30558.67	0.00%	42279.13	0.00%	29658.17	0.00%	158384.22	0.03%	73159.18	0.01%	2h 45min
ORTools	31072.05	1.68%	43678.57	3.31%	30752.55	3.69%	165319.37	4.41%	75390.32	3.06%	14h 46min
DIFUSCO (Ts=50)	30830.64	0.89%	43336.11	2.50%	30758.49	3.71%	-	-	-	-	-
SGBS	30803.14	0.80%	45187.94	6.88%	34750.47	17.17%	235035.03	48.44%	86670.34	18.48%	6h 29min
CNF (3)	30867.31	1.01%	46266.05	9.43%	38119.65	28.53%	251882.05	59.08%	90284.04	23.42%	1h 18min
BQ (greedy)	31270.68	2.33%	43623.61	3.18%	32122.76	8.31%	225740.66	42.57%	84614.77	15.67%	13min
BQ (bs16)	30907.04	1.14%	43200.82	2.18%	31295.30	5.52%	216493.80	36.73%	82603.09	12.92%	4h 7min
ELG (no aug)	31255.40	2.28%	44985.00	6.40%	32398.58	9.24%	178508.82	12.74%	78309.08	7.05%	7min
ELG ($\times$8 aug)	30910.09	1.15%	43961.84	3.98%	32247.32	8.73%	176323.77	11.36%	77263.00	5.62%	39min
POMO (no aug)	32318.85	5.76%	48481.48	14.67%	38104.81	28.48%	272069.99	71.83%	95375.41	30.38%	-
POMO ($\times$8 aug)	31973.53	4.63%	46701.53	10.46%	37461.23	26.31%	261002.25	64.84%	92654.16	26.66%	26min
LEHD (greedy)	31179.01	2.03%	43441.81	2.75%	30859.32	4.05%	176181.27	11.27%	76999.66	5.26%	6min
LEHD (RRC-50)	30659.51	0.33%	42558.18	0.66%	30289.89	2.13%	168581.11	6.47%	74966.04	2.48%	1h 51min
POMO (ours): no aug	30946.76	1.27%	43192.36	2.16%	31366.48	5.76%	213754.57	35.00%	82259.28	12.45%	-
POMO (ours): $\times$8 aug	30907.04	1.14%	43192.36	2.16%	31366.48	5.76%	209685.32	32.43%	81630.17	11.59%	26min
LEHD (ours): greedy	30986.49	1.40%	43116.26	1.98%	30666.54	3.40%	166839.40	5.37%	75302.53	2.94%	6min
LEHD (ours): RRC-50	**30650.34**	**0.30%**	**42427.11**	**0.35%**	**29874.67**	**0.73%**	**162358.47**	**2.54%**	**73919.96**	**1.05%**	1h 52min

Table 1: Performance comparison on 70 TSPLIB problems. Each entry shows the range-wise average objective and optimality gap (%). along with the total inference time in the last column. The best results of all learning-based methods are in **bold**.

a greater number of training epochs. Additionally, we continuously assess model performance throughout phase one using the 48 validation problems.

Phase Two. After aligning the neural model with a broader range of structural priors in phase one, we fine-tune the best-performed model saved in phase one with TSPLib instances and validate the model on themselves. This enables the model to shift from easy tasks (synthetic data) to accommodate to hard cases (real-world instances) seamlessly, forming a synthetic-real progression. By combining both phases together, our strategy not only enables models to generalize from synthetic to real distributions, but also goes beyond standard curriculum or domain adaptation by sequentially bridging both distributional and scale gaps.

4 Experimental Results

In this section, we evaluate EvoReal on TSPLib95 (Reinelt 1991) and CVRPLib setX (Uchoa et al. 2017).

Experimental Settings

Baselines. We employ four classic solvers as non-neural baselines: Concorde (Applegate et al. 2006), ORTools, HGS (Vidal 2022) and LKH-3 (Helsgaun 2017). For deep neural solvers, POMO (Kwon et al. 2020), LEHD (Luo et al. 2023), SGBS (Choo et al. 2022), BQ (Drakulic et al. 2023), ELG (Gao et al. 2024), CNF (Zhou et al. 2024) and DIFUSCO (Sun and Yang 2023). Importantly, POMO is trained using reinforcement learning (RL) with policy gradient methods, whereas LEHD employs a supervised learning (SL) scheme that leverages optimal objective values. These models sample solely uniform distributions for training.

Training Setups. Hyperparameters have two sets: one for proxy evaluation and the other for progressive fine-tuning. We choose two representative attention-based neural models for our study, namely POMO and LEHD. For both sets, we leverage the evolved generators to produce TSP or CVRP of size 100 for fine-tuning. In phase two, instances from VRPLib are expanded by a pre-set batch size as a single batch, and they are expanded only once in one epoch. For LEHD, Concorde solver and HGS are adopted to obtain the optimum of TSP and CVRP, respectively. For generator evolution, we use OpenAI model o3 to construct new generators. All evolution pipelines can be conducted on a single NVIDIA RTX 3090 Ti GPU with 24GB memory, whereas the progressive fine-tuning is performed on one NVIDIA RTX A6000 GPU with 48GB memory. The complete hyperparameters are given in section D of the extended version.

Metric and Inference. We compare the results with different metrics with respect to different problem size intervals, including average objective values, average gaps (to the optimal average tour length), and total inference time. In order to monitor the performance of models during the proxy evaluation and progressive fine-tuning, we evaluate the models' performance at fixed epochs. At each validation point, we calculate the total average gap on the validation set as the performance metric with 8-fold augmentation for POMO or greedy mode for LEHD. The inference time of classic non-neural solvers (CPU) and learning-based

Method	[0,200)		[200,500)		[500,1002)		Overall		Time
	Obj	Gap	Obj	Gap	Obj	Gap	Obj	Gap	
Count	22		46		32		100		
HGS-CVRP	27222.86	0.01%	53334.09	0.06%	102118.28	0.24%	63176.43	0.11%	16h 40min
LKH-3	27315.41	0.35%	53845.79	1.02%	103310.20	1.41%	63738.08	1.00%	27h 29min
ORTools	27802.65	2.14%	55514.15	4.15%	106987.85	5.02%	65637.60	4.01%	8h 4min
CNF (3)	28964.95	6.41%	61244.12	14.90%	143774.47	41.13%	76624.53	21.42%	10min 34s
ELG (no aug)	28921.39	6.25%	57342.41	7.58%	112081.53	10.02%	68199.75	8.07%	2m 15s
ELG (×8 aug)	28447.76	4.51%	56244.39	5.52%	109819.94	7.80%	66912.36	6.03%	5m 24s
POMO (no aug)	29822.38	9.56%	63685.36	19.48%	161266.20	58.30%	81862.41	29.72%	-
POMO (×8 aug)	29057.50	6.75%	61270.77	14.95%	143794.84	41.15%	76693.95	21.53%	2m 17s
LEHD (greedy)	30309.62	11.35%	58339.16	9.45%	119946.19	17.74%	71008.01	12.52%	2m 18s
LEHD (RRC-50)	28556.65	4.91%	55823.30	4.73%	110298.74	8.27%	66830.32	5.90%	1h 10min
POMO (ours): no aug	28279.00	3.89%	55940.56	4.95%	108505.76	6.51%	66401.20	5.22%	-
POMO (ours): ×8 aug	28178.29	3.52%	55754.01	4.60%	108189.96	6.20%	66180.32	4.87%	2m 6s
LEHD (ours): greedy	28782.57	5.74%	55700.70	4.50%	106172.85	4.22%	66060.42	4.68%	2m 36s
LEHD (ours): RRC50	**27916.97**	**2.56%**	**54682.63**	**2.59%**	**104929.99**	**3.00%**	**64817.21**	**2.71%**	1h 9min

Table 2: Performance comparison over 100 SetX problems, with average objective, optimality gap (%), and inference time. The best result of all learning-based methods under each problem size range are marked in **bold**.

POMO (aug×8)	[0,200)	[200,1000)	[1000,5000)	Time
w/o Fine-tune	4.63%	14.99%	64.84%	-
w/o Phase 1	1.63%	12.06%	44.85%	7h 41min
w/o Phase 2	**1.10%**	11.14%	43.06%	5h 42min
Full (ours)	1.11%	**6.78%**	**36.48%**	6h 37min

Table 3: Comparison of different POMO fine-tuning setups. We use problem of nodes ≤ 500 with batch size 4 in TSPLib for finetuning. All experiments are trained for 600 epochs. The best result for each problem range is in **bold**.

LEHD - greedy	[0,200)	[200,1000)	[1000,5000)	Time
w/o Fine-tune	2.03%	3.12%	11.27%	-
w/o Phase 1	2.55%	3.24%	8.7%	31 min
w/o Phase 2	1.97%	**2.35%**	6.08%	35 min
Full (ours)	**1.40%**	2.39%	**5.37%**	38.5 min

Table 4: Comparison of different LEHD fine-tuning setups. We use problem of nodes ≤ 500 with batch size 4 in TSPLib for finetuning. All experiments are trained for 40 epochs. The best result for each problem range are in **bold**.

models (GPU) are not directly comparable. For POMO, LEHD, SGBS, CNF(3), ELG, DIFUSCO (Ts=50) we report the performance of their pre-trained models with their default settings. For BQ, we reproduced their model with their original setups. For the classic non-neural solvers for TSP, we set a maximum time limit according to problem size n ($T_{max} = 1.8 * n$ seconds for LKH-3 and ORTools, and $T_{max} = 10$ minutes for Concorde). As for non-neural solvers for CVRP, we cite the results from Table 2 of HGS paper (Vidal 2022). For POMO, we provide the results with and without 8 times augmentation (×8 AUG and non aug). With respect to LEHD, the inference results under both greedy mode and RRC (with 50 iterations) are documented.

Main Results

The main experimental results for TSPLib and CVRPLib are detailed in Table 1 and Table 2. As shown in Table 1, we observed that compared to recent SOTA baselines, our method, namely POMO (ours) and LEHD (ours) both significantly reduce objective values and optimal gaps across all problem sizes. Markedly, with 50 RRC iterations, LEHD (ours) dominates all other learning-based methods and ORTools across all sizes. Even compared to LEHD (RRC-50) itself, most gap reductions are more than a half, without notably increasing the inference time. POMO (ours) also outstrips solvers like SGBS, BQ and POMO, especially on large instances, both with and without augmentation. Compared to the pre-trained POMO, the gaps also dropped by more than a half. Given that the model architectures for POMO and LEHD remain intact, the inference time is only subject to trivial variance. POMO (non-augmentation) and LEHD (greedy) exhibit the lowest and second-lowest inference costs, respectively, while the costs associated with their augmented variants remain within a manageable range.

Regarding the generalization in CVRPLib, we report the same metrics as before in Table 2. Similarly to their TSP counterparts, POMO (ours) and LEHD (ours) under both non-augmented and augmented versions have witnessed consistent performance improvement, particularly in large-size instances. With 50 RRC iterations, LEHD under pro-

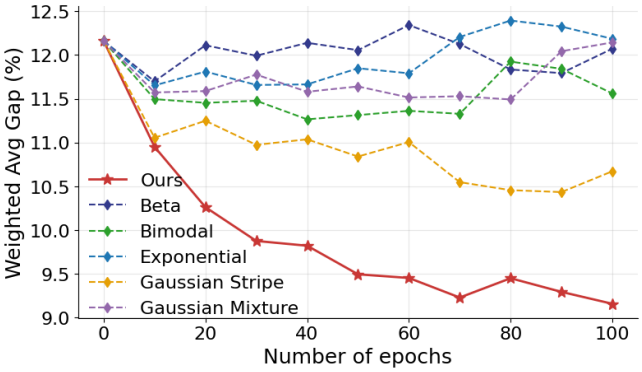

Figure 3: Comparison of the performance of the evolved generator with five the naive-distribution generators.

gressive fine-tuning has outperformed all neural solvers, and the results surpass ORTools in other problem intervals whereas it is still competitive against it with 0-200 nodes. The total average gaps of ours are comparable in all problem ranges (of the largest gap difference being 2.68%), indicating that progressive fine-tuning has significantly narrowed the performance difference between small-size instances and large-size ones. POMO under progressive fine-tuning also demonstrates improved performance over ELG and the original POMO across all problem sizes. It is worth noting that LEHD (ours) show even smaller average gap in large-size instances compared to small-size ones, possibly due to overfitting on small-size problems in the second phase of progressive fine-tuning. Additionally, all inference times vary from those of pre-trained models at a marginal level. Table 6 in the extended version also shows that with our framework, neural solvers can also generalize well on the problems unseen during training.

In summary, our EvoReal framework has substantially enhanced the models' generalization capability on VRPLib, particularly on large problems. Ablation studies in Table 3 and Table 4 further corroborate the efficacy of our method.

5 Ablation Studies

Phase Level Ablation. We investigate the impact of each phase in progressive fine-tuning. Under two pre-trained backbone models for TSP, we ablate phase one and phase two separately, and report the range-wise average gaps and the total training time of the full progressive fine-tuning (Full), two ablation setups (w/o Phase 1 and w/o Phase 2) as well as not fine-tuning at all (w/o fine-tune). As shown in Table 3, the gaps in our full framework consistently outperform other setups excluding problem size [0,200], in which the gap marginally degrades 0.01% compared to without phase 2. Furthermore, compared to without phase 1, adding phase 1 markedly contributes to the performance improvement in problem size [200,1000) and [1000,5000), while the performance of without phase 1 is limited by the diversity of training instances. Similarly in Table 4, for LEHD-greedy, our full framework is superior on both small-size and large-size problems, with only minor degradation (at 0.04%) on the medium-size one. This further confirms that our progressive design can synergistically narrow the gap while maintaining an acceptable training cost. Fig.6 in the extended version displays the training curves corresponding to Table 3 and 4.

Method	Avg. Aug Gap (%)
W/O Finetuning	22.21 ±0.00
W/O LLM	19.79 ±0.25 (best: 19.46)
W/O Rank-based Selection	16.88 ±0.06 (best: 16.79)
W/O Design Guidance	16.62 ±0.05 (best: 16.56)
W/O Late Selection	16.55 ±0.07 (best: 16.50)
Full EvoReal	**16.52** ±0.05 (best: **16.41**)

Table 5: Ablation of key mechanisms in our evolution framework. Each entry shows the average fitness score of top-5 generators, with standard deviation and the score of best one.

Comparison with Naive Generators. We also compare our evolved three TSP generators, combined into a single generator, with five naive distribution generators, including beta, exponential, binomial, Gaussian mixture, and Gaussian stripe. All generators fine-tune the pre-trained POMO TSP model for 100 epochs. The trends of the total average gap in Fig.3 indicate that our evolved generator with mixed distribution exceeds all naive distribution generators from the outset and ultimately converges at a much lower average gap, substantiating that our evolved generator better captures the structural characteristics of TSPLib distributions.

LLM Mechanism-level Ablation To verify the usefulness of the modified components in our LLM-driven evolution framework, we perform the generator evolution on S1-type generator under different setups, including the removal of rank-based selection, the removal of the design-guidance prompt, and advancing the selection before evolution. We use the manually-crafted seed generator to fine-tune the pre-trained model for w/o LLM experiment. The performance of the original model is also added for comparison (w/o fine-tuning). The average fitness score of the top five elitist generators, along with the standard deviation and the fitness score of the best generator in each setup, are reported in Table 5.

6 Conclusion and Future Works

This work introduces a novel LLM-guided evolution framework for evolving VRP data distributions, combined with a progressive fine-tuning strategy to progressively adapt neural solvers from synthetic to diverse real-world instances. Our approach enables automated discovery and evolution of real-world-aligned VRP data generators, bridging the generalization gap for two neural solvers. Experimental results on two benchmarks demonstrate that the models trained with our evolved structural-aligned generators significantly outperform strong baselines, particularly on large-size instances, without requiring architectural changes or costly retraining. For future work, we will extend the LLM-driven evolution framework to other non-VRP tasks, such as MIS and bin packing, which involve richer structural and combinatorial constraints requiring expressive generator designs.

Acknowledgments

This research is supported by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031, and AISG-NMLP-2024-003).

References

Applegate, D.; Bixby, R.; Chvatal, V.; and Cook, W. 2006. Concorde TSP solver.

Bdeir, A.; Falkner, J. K.; and Schmidt-Thieme, L. 2022. Attention, filling in the gaps for generalization in routing problems. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 505–520. Springer.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural Combinatorial Optimization with Reinforcement Learning. arXiv:1611.09940.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Bi, J.; Ma, Y.; Wang, J.; Cao, Z.; Chen, J.; Sun, Y.; and Chee, Y. M. 2022. Learning Generalizable Models for Vehicle Routing Problems via Knowledge Distillation. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *NeurIPS*.

Cappart, Q.; Chételat, D.; Khalil, E. B.; Lodi, A.; Morris, C.; and Veličković, P. 2023. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130): 1–61.

Cattaruzza, D.; Absi, N.; Feillet, D.; and González-Feliu, J. 2017. Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, 6(1): 51–79.

Chalumeau, F.; Surana, S.; Bonnet, C.; Grinsztajn, N.; Pretorius, A.; Laterre, A.; and Barrett, T. 2023. Combinatorial optimization with policy adaptation using latent space search. *NeurIPS*, 36: 7947–7959.

Chen, J.; Zhang, Z.; Cao, Z.; Wu, Y.; Ma, Y.; Ye, T.; and Wang, J. 2023. Neural multi-objective combinatorial optimization with diversity enhancement. In *NeurIPS*, volume 36, 39176–39188.

Choo, J.; Kwon, Y.-D.; Kim, J.; Jae, J.; Hottung, A.; Tierney, K.; and Gwon, Y. 2022. Simulation-guided beam search for neural combinatorial optimization. *NeurIPS*, 35: 8760–8772.

Drake, J. H.; Kheiri, A.; Özcan, E.; and Burke, E. K. 2020. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2): 405–428.

Drakulic, D.; Michel, S.; Mai, F.; Sors, A.; and Andreoli, J.-M. 2023. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *NeurIPS*, 36: 77416–77429.

Duflo, G.; Kieffer, E.; Brust, M. R.; Danoy, G.; and Bouvry, P. 2019. A gp hyper-heuristic approach for generating tsp heuristics. In *IPDPSW*, 521–529. IEEE.

Elhenawy, M.; Abdelhay, A.; Alhadidi, T. I.; Ashqar, H. I.; Jaradat, S.; Jaber, A.; Glaser, S.; and Rakotonirainy, A. 2024a. Eyeballing combinatorial problems: A case study of using multimodal large language models to solve traveling salesman problems. In *International Conference on Intelligent Systems, Blockchain, and Communication Technologies*, 341–355. Springer.

Elhenawy, M.; Abutahoun, A.; Alhadidi, T. I.; Jaber, A.; Ashqar, H. I.; Jaradat, S.; Abdelhay, A.; Glaser, S.; and Rakotonirainy, A. 2024b. Visual Reasoning and Multi-Agent Approach in Multimodal Large Language Models (MLLMs): Solving TSP and mTSP Combinatorial Challenges. *Machine Learning and Knowledge Extraction*, 6(3): 1894–1920.

Gao, C.; Shang, H.; Xue, K.; Li, D.; and Qian, C. 2024. Towards Generalizable Neural Solvers for Vehicle Routing Problems via Ensemble with Transferrable Local Policy. In *IJCAI*, 6914–6922.

Goh, Y. L.; Ma, Y.; Zhou, J.; Cao, Z.; Dupty, M. H.; and Lee, W. S. 2025. SHIELD: Multi-task Multi-distribution Vehicle Routing Solver with Sparsity & Hierarchy in Efficiently Layered Decoder.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12: 966–980.

Hottung, A.; Mahajan, M.; and Tierney, K. 2024. PolyNet: Learning Diverse Solution Strategies for Neural Combinatorial Optimization.

Hou, Q.; Yang, J.; Su, Y.; Wang, X.; and Deng, Y. 2023. Generalize Learned Heuristics to Solve Large-scale Vehicle Routing Problems in Real-time. In *ICLR*.

Huang, Y.; Zhang, W.; Feng, L.; Wu, X.; and Tan, K. C. 2024. How Multimodal Integration Boost the Performance of LLM for Optimization: Case Study on Capacitated Vehicle Routing Problems. *CoRR*, abs/2403.01757.

Iklassov, Z.; Du, Y.; Akimov, F.; and Takáč, M. 2024. Self-Guiding Exploration for Combinatorial Problems. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *NeurIPS*, volume 37, 130569–130601. Curran Associates, Inc.

Jiang, X.; Wu, Y.; Li, M.; Cao, Z.; and Zhang, Y. 2025a. Large Language Models as End-to-end Combinatorial Optimization Solvers. In *NeurIPS*.

Jiang, X.; Wu, Y.; Zhang, C.; and Zhang, Y. 2025b. DRoC: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints. In *ICLR*.

Joshi, C.; Cappart, Q.; Rousseau, L.; Laurent, T.; and Bresson, X. 2006. Learning TSP requires rethinking generalization. arXiv 2020. *arXiv preprint arXiv:2006.07054*.

Kim, M.; Park, J.; and Park, J. 2022. Sym-NCO: Leveraging Symmetricity for Neural Combinatorial Optimization. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *NeurIPS*, volume 35, 1936–1949. Curran Associates, Inc.

Kim, M.; Son, J.; Kim, H.; and Park, J. 2022. Scale-conditioned Adaptation for Large Scale Combinatorial Optimization. In *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*.

Konstantakopoulos, G. D.; Gayialis, S. P.; and Kechagias, E. P. 2022. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, 22(3): 2033–2062.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *ICLR*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *NeurIPS*, volume 33, 21188–21198. Curran Associates, Inc.

Kwon, Y.-D.; Choo, J.; Yoon, I.; Park, M.; Park, D.; and Gwon, Y. 2021. Matrix encoding networks for neural combinatorial optimization. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *NeurIPS*, volume 34, 5138–5149. Curran Associates, Inc.

Li, S.; Yan, Z.; and Wu, C. 2021. Learning to delegate for large-scale vehicle routing. *NeurIPS*, 34: 26198–26211.

Lin, Z.; Wu, Y.; Zhou, B.; Cao, Z.; Song, W.; Zhang, Y.; and Jayavelu, S. 2024. Cross-Problem Learning for Solving Vehicle Routing Problems. In *IJCAI*, 6958–6966.

Lisicki, M.; Afkanpour, A.; and Taylor, G. W. 2020. Evaluating Curriculum Learning Strategies in Neural Combinatorial Optimization. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Liu, F.; Xialiang, T.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; and Zhang, Q. 2024a. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *ICML*.

Liu, S.; Chen, C.; Qu, X.; Tang, K.; and Ong, Y.-S. 2024b. Large Language Models as Evolutionary Optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Liu, S.; Zhang, Y.; Tang, K.; and Yao, X. 2023. How Good is Neural Combinatorial Optimization? A Systematic Evaluation on the Traveling Salesman Problem. *IEEE Computational Intelligence Magazine*, 18(3): 14–28.

Luo, F.; Lin, X.; Liu, F.; Zhang, Q.; and Wang, Z. 2023. Neural Combinatorial Optimization with Heavy Decoder: Toward Large Scale Generalization. In *NeurIPS*.

Ma, Y.; Cao, Z.; and Chee, Y. M. 2023. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *NeurIPS*, 36: 49555–49578.

Nazari, M.; Oroojlooy, A.; Snyder, L.; and Takac, M. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *NeurIPS*, volume 31. Curran Associates, Inc.

Ramji, K.; Lee, Y.-S.; Astudillo, R. F.; Sultan, M. A.; Naseem, T.; Munawar, A.; Florian, R.; and Roukos, S. 2024. Self-Refinement of Language Models from External Proxy Metrics Feedback. *CoRR*, abs/2403.00827.

Reinelt, G. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384.

Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.

Son, J.; Kim, M.; Kim, H.; and Park, J. 2023. Meta-SAGE: Scale Meta-Learning Scheduled Adaptation with Guided Exploration for Mitigating Scale Shift on Combinatorial Optimization. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *ICML*, volume 202 of *Proceedings of Machine Learning Research*, 32194–32210. PMLR.

Sun, Z.; and Yang, Y. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *NeurIPS*, 36: 3706–3731.

Tran, C. D.; Nguyen-Tri, Q.; Binh, H. T. T.; and Thanh-Tung, H. 2025. Large Language Models powered Neural Solvers for Generalized Vehicle Routing Problems. In *Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*.

Uchoa, E.; Pecin, D.; Pessoa, A.; Poggi, M.; Vidal, T.; and Subramanian, A. 2017. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3): 845–858.

Vidal, T. 2022. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140: 105643.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *NeurIPS*, volume 28. Curran Associates, Inc.

Wang, H.; Feng, S.; He, T.; Tan, Z.; Han, X.; and Tsvetkov, Y. 2023. Can Language Models Solve Graph Problems in Natural Language? In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *NeurIPS*, volume 36, 30840–30861. Curran Associates, Inc.

Xu, X.; Xiao, T.; Chao, Z.; Huang, Z.; Yang, C.; and Wang, Y. 2025. Can LLMs Solve Longer Math Word Problems Better? In *ICLR*.

Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q. V.; Zhou, D.; and Chen, X. 2024. Large Language Models as Optimizers. In *ICLR*.

Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; and Song, G. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. *NeurIPS*, 37: 43571–43608.

Zhou, J.; Wu, Y.; Cao, Z.; Song, W.; Zhang, J.; and Shen, Z. 2024. Collaboration! Towards Robust Neural Methods for Routing Problems. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *NeurIPS*, volume 37, 121731–121764. Curran Associates, Inc.

Zhou, J.; Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2023. Towards Omni-generalizable Neural Methods for Vehicle Routing Problems. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *ICML*, volume 202 of *Proceedings of Machine Learning Research*, 42769–42789. PMLR.