SCALING AUTOREGRESSIVE MODELS FOR ROUTING-STYLE NEURAL COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive (AR) models have long served as a flexible and general paradigm for neural combinatorial optimization (NCO), naturally constructing feasible solutions through sequential decisions. However, scaling AR solvers to larger instances remains challenging, especially under Reinforcement Learning (RL), where long horizons, sparse rewards, and expensive decoding limit both efficiency and performance. We argue that this limitation is not inherent to autoregression, but arises because existing AR designs do not jointly support expressive decision modeling, efficient incremental decoding, and stable long-horizon training. We propose PrefixCO, a prefix-autoregressive paradigm for scaling AR models for routing-style NCO. PrefixCO encodes problem instance attributes into reusable prefix representations and causally generates solutions with cached key-value states, computing static instance information once while allowing each step to condition on the evolving partial solution. This preserves AR flexibility and reduces the average per-step decoding complexity to $O(N)$ from $O(N^2)$. PrefixCO further adopts a cache-compatible pointer decoder that matches prefix-derived candidate-node keys with a trajectory-conditioned query, producing a step-specific distribution over feasible next nodes while preserving incremental decoding efficiency. For training, PrefixCO introduces a scalable RL protocol that combines imitation warm-starting with non-negative improvement-based reward shaping. Experiments on TSP and CVRP show that PrefixCO establishes a strong quality-efficiency trade-off: under RL, it reduces the optimality gap on 1000-node TSP by **30**$\times$ over prior AR baselines and achieves a **2**$\times$ reduction on CVRP.

1 INTRODUCTION AND RELATED WORKS

Combinatorial Optimization (CO) serves as the foundation for numerous real-world applications, including vehicle routing (Veres & Moussa, 2020), drug discovery (Liu et al., 2017), and chip design (Cheng et al., 2022). Neural Combinatorial Optimization (NCO) has emerged as a promising paradigm, leveraging machine learning to automate the design of heuristics. Within learning-based solvers, the solution generation paradigm is primarily categorized into autoregressive and non-autoregressive approaches. Autoregressive (AR) methods construct solutions through sequential decisions (Kool et al., 2018; Kwon et al., 2020; Pan et al., 2025a; Huang et al., 2025; Luo et al., 2024; Drakulic et al., 2023; Kim et al., 2022), while non-autoregressive (NAR) methods typically predict global solution structures in a fixed number of steps, often guided by heatmaps or generative models (Joshi et al., 2019; Qiu et al., 2022; Li et al., 2024; Sanokowski et al., 2024b; Li et al., 2025b;b; Chen et al., 2026). Both paradigms have shown strong progress. Non-autoregressive models offer efficient parallel prediction, whereas autoregressive models provide a flexible and general formulation that naturally

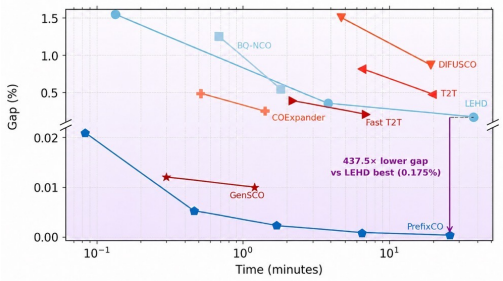

Figure 1: Performance comparison of AR solvers (blue) and NAR solvers (orange) on TSP500. PrefixCO extends AR solvers to a new scale regime.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

supports feasibility constraints, partial solutions, and diverse CO variants (Zong et al., 2025; Zhou et al., 2024b; Drakulic et al., 2025; Pan et al., 2025b; Li et al., 2025c).

Despite their flexibility, scaling AR to larger instances remains an open question. The difficulty is especially evident under Reinforcement Learning (RL) (Kool et al., 2018; Kwon et al., 2020; Pan et al., 2025a), where the decision horizon grows with the problem size, terminal rewards are sparse, and high-throughput rollout generation is required for effective policy optimization. Meanwhile, AR decoding can become computationally expensive when it must repeatedly recompute instance representations during solution construction. These challenges can limit both efficiency and performance, making it difficult for AR solvers to benefit from larger models, larger training budgets, and larger problem sizes.

We argue that this limitation is not inherent to autoregression itself. Rather, it arises because *existing AR designs do not jointly support three requirements that are crucial for scaling: expressive decision modeling, efficient incremental decoding, and stable long-horizon training*. Classical AR methods, such as the Attention Model (Kool et al., 2018), employ an encoder to process static node features and a lightweight decoder to select the next node based on the partial solution and global context. This design enables efficient decoding because static key-value (KV) representations can be reused, but the lightweight decoder may limit the model's ability to capture rich interactions between candidate nodes, the global instance structure, and the evolving solution state. In contrast, more expressive AR methods e.g. BQ-NCO (Drakulic et al., 2023) and LEHD (Luo et al., 2024) adopt multi-layer decoders to model dynamic decision states, yet this often prevents efficient caching and requires repeated computation at each decoding step. Hence, they may incur an average complexity of $O(N^2)$ per step and $O(N^3)$ to construct a complete solution.

Recent prefix-style architectures provide an initial step toward scaling AR solvers. The CO prefix architecture (Zong et al., 2025) reformulates solution generation in a language-model-like manner: static problem information is placed in a prefix, and solutions are generated causally as a trajectory. By separating static instance processing from causal solution generation, this design enables KV caching and reduces autoregressive decoding to an average complexity of $O(N)$ per step. However, existing prefix-style solvers have not yet realized the full scaling potential of AR-NCO. *A fundamental reason is that existing designs largely inherit the standard LLM paradigm, whereas combinatorial optimization differs from language modeling in several crucial aspects*: 1) CO instances are often defined by continuous numerical attributes, and small numerical perturbations can lead to substantially different optimal solutions. Thus, directly discretizing continuous attributes into tokens may introduce harmful errors. 2) The action space in routing-style CO is instance-specific and heavily constrained: each step must select from feasible candidate nodes rather than a fixed vocabulary, requiring decoding to explicitly score valid candidates conditioned on both the static instance and the evolving partial solution. 3) Scaling prefix-style AR solvers under RL is more delicate, as long horizons, sparse rewards, and feasibility constraints make rollout-based optimization unstable. These differences make a direct transfer of LLM techniques insufficient for scaling AR-NCO.

To scale AR solvers for routing-style NCO, we need a CO-aware prefix-autoregressive paradigm that preserves cached incremental decoding while respecting the numerical, constrained, and long-horizon nature of combinatorial decision making. To this end, we propose PrefixCO, a unified scaling recipe organized around three requirements: expressive decision modeling, efficient incremental decoding, and stable long-horizon training. Architecturally, PrefixCO encodes continuous instance attributes into reusable prefix representations, avoiding discrete tokenization and preserving the numerical structure of CO instances. It then generates solutions causally with cached KV states, so static instance information is computed once while each decoding step still conditions on the evolving partial solution. For action prediction, PrefixCO adopts a cache-compatible pointer decoder within the Transformer architecture: prefix outputs are projected into candidate-node keys, the current trajectory state is projected into a query, and normalized key-query similarities define a step-specific distribution over feasible next nodes. This design captures candidate-node, instance-level, and solution-state interactions while retaining $O(N)$ average per-step decoding.

For training, PrefixCO further introduces a scalable RL protocol tailored to long-horizon AR-NCO. Directly applying POMO-style policy optimization to large prefix-autoregressive models can be unstable: randomly initialized policies produce uninformative rollouts, and raw cost-based rewards can lead to high-variance updates or degradation of the policy distribution. We address this with imitation warm-starting and non-negative improvement-based reward shaping. Imitation warm-starting moves the policy into a region where generated trajectories are already meaningful, making

subsequent RL feedback informative. Non-negative reward shaping converts raw costs into instance-wise improvement signals, reducing the influence of uninformative negative trajectories and stabilizing policy optimization. Together with cache-efficient decoding, this training protocol allows additional RL compute to translate into reliable performance gains on larger instances.

Empirically, our redesigned AR framework brings concrete improvements to AR solvers, achieving state-of-the-art results with a markedly better quality-efficiency trade-off. On TSP, it attains near-optimal solution quality and reduces the optimality gap on large-scale instances by **30×** over prior SOTA AR baselines. On CVRP, it also delivers substantial gains achieving a **2×** reduction on the optimality gap. Moreover, under supervised learning, it surpasses non-AR solvers on large-scale TSP with an **7×** gap reduction, without post-processing heuristics.

2 Preliminaries and Problem Definition

Combinatorial Optimization. A CO problem instance is defined over a graph $G = (V, E)$, where $v_i \in V = \{v_i\}_{i=1}^{N}$ represents the node and $e(v_i, v_j) \in E$ represents the edge between node v_i and v_j. A feasible solution τ is represented as a node sequence that satisfies the problem-specific constraints.

Autoregressive Formulation. A solution $\tau = (\tau_1, \tau_2, \ldots, \tau_T)$ can be generated by policy p_θ:

$$p_\theta(\tau \mid G) = \prod_{t=2}^{T} p_\theta(\tau_t \mid \tau_{1:t-1}, G), \tag{1}$$

where T is the total number or construction steps, τ_t represents the node selected at step t, $\tau_{1:t-1}$ represents the partial solution constructed up to step $t-1$, and $p_\theta(\tau_t \mid \tau_{1:t-1}, G)$ denotes the probability of selecting the next node given the current state. The construction process terminates when a complete feasible solution τ is formed.

Supervised Learning Objective. When a ground-truth solution τ^* is available for an instance G, the supervised learning (SL) objective is to minimize the cross-entropy loss between the predicted node distribution and the choices of the reference policy to imitate:

$$\mathcal{L}_{SL}(\theta) = -\log p_\theta(\tau^* \mid G), \tag{2}$$

This objective encourages the model to maximize the likelihood of the reference policy's decisions at each construction step. Note that the solutions are not necessary to be optimal or even high-quality.

Reinforcement Learning Objective. The policy is trained to minimize the expected cost $c(\tau)$ directly. In POMO (Kwon et al., 2020)-like group-based algorithms, for a group of sampled trajectories $\{\tau^1, \ldots, \tau^M\}$ (where M is the group size) of **a single problem instance** and a reward $R(\cdot)$, the objective is defined:

$$\mathcal{L}_{RL}(\theta) = -\frac{1}{M} \sum_{i=1}^{M} \widehat{R}(\tau^i \mid \{\tau^j\}_{j=1}^{M}) \log p_\theta(\tau^i \mid G), \tag{3}$$

where $\widehat{R}(\cdot \mid \cdot)$ is the group-based shaped reward function. For example, in POMO, $\widehat{R}(\tau^i \mid \{\tau^j\}_{j=1}^{M}) = R(\tau^i) - \frac{1}{M}\sum_{j=1}^{M} R(\tau^j)$.

Represent Instance as Prefix. Static node features can be transformed into prefix by an embedder, which can be a discrete tokenizer, a linear projection, or even stacked transformer layers. Denote the prefix sequence as P, the separator token as $\langle \mid \rangle$. At each autoregressive step, decision on τ_i is conditioned on $(P; \langle \mid \rangle; \tau_1, \ldots, \tau_{i-1})$.

3 Methodology

For the design of PrefixCO, Sec. 3.1 describes the prefix-transformer architecture and its adaptations to the numerical structure, instance-specific candidate sets, and dynamic constraints of CO problems. Sec. 3.2 introduces a reward-shaping scheme resolving the identified training-instability specific to prefix transformers. Sec. 3.3 outlines two complementary techniques that complete the training recipe, grouped-query attention and imitation warm-starting, with further details in Appendix B.

3.1 PREFIX TRANSFORMER FOR CO

Generation Procedure. The prefix sequence P is first processed by the decoder in a single pass to compute and store the KV cache. During this initial phase, the attention mechanism is fully bidirectional, allowing each token within the prefix to attend to all others. In the subsequent generation stage, the model follows standard autoregressive decoding: the current token can only attend to the stored prefix and the nodes generated before it, which is identical to the behavior of a decoder-only Transformer. This hybrid approach ensures that the prefix provides a rich, global context while maintaining the efficiency of the causal decoder during sequence construction.

Trajectory Components. In Zong et al. (2025), each step of a trajectory comprises multiple tokens representing both an action and a state. However, since the state component does not introduce critical information beyond what is already provided by the prefix, we retain only the action token to reduce token consumption. Consequently, each trajectory step now corresponds strictly to a single action token.

Embedder. In Zong et al. (2025), the it is implemented as a discrete tokenizer mapping continuous and discrete values into 1,800 and 200 tokens, respectively. While this simplified the input pipeline, it introduced significant rounding errors, which is a critical drawback for CO, being often sensitive to small numerical fluctuations. Furthermore, the flattening of node features to accommodate the tokenizer resulted in high sequence overhead. For a TSP instance with N nodes, the prefix consumes $2N$ tokens; for a CVRP, the cost increases to $3N + 2$ tokens. We adopt a more direct approach by treating all features as continuous values. Specifically, we employ a linear projection as the default embedder, which can be substituted with more complex, task-specific architectures if necessary. The benefit of this choice grows with problem size: within the same coordinate space, larger instances place nodes more densely, amplifying the rounding errors of discrete tokenization. A controlled ablation against a discrete tokenizer on the same backbone is given in Table 3.

Output Layer. By design of the embedder, each prefix token corresponds strictly to a node, enabling a **pointer network (Vinyals et al., 2015)-style** decoding mechanism, as illustrated in Fig. 2. Specifically, the outputs from the prefix part are linearly projected to serve as *decoding keys*, while the outputs from the generation part are linearly projected to form *decoding queries*. For each decoding query, we compute dot products with all available decoding keys, followed by a softmax operation to produce a probability distribution over the candidate next nodes, where each node corresponds to a decoding query. Notably, the projection parameters for queries and keys are not shared. This is because the decoding keys represent features of the current (input) nodes, whereas the decoding queries must encode information relevant to the *next* node in the sequence. Consequently, even when using the same backbone model, distinct linear projections are required to map these representations into separate semantic spaces appropriate for their respective roles. Formally, let n denote the length of the prefix, d the model dimension, $H \in \mathbb{R}^{n\times d}$ the output of the prefix sequence, and $x \in \mathbb{R}^d$ the output of the current query node. Given the decoding query projection $W^Q \in \mathbb{R}^{d\times d}$ and the decoding key projection $W^K \in \mathbb{R}^{d\times d}$, the next node probability distribution p is defined as:

$$p = \mathrm{softmax}(\mathrm{Norm}(HW^K) \cdot \mathrm{Norm}(xW^Q)^\top). \tag{4}$$

The $\mathrm{Norm}(\cdot)$ function denotes RMSNorm (Zhang & Sennrich, 2019), which is applied to the projected representations to ensure numerical stability and prevent gradient exploding/vanishing issues during the dot-product operation. To maintain the integrity of the decoding process, invalid candidate nodes are excluded either by temporarily removing them from the prefix matrix H or by applying a mask to the pre-softmax logits. The resulting vector $p \in \mathbb{R}^n$ represents the normalized scores, where each element corresponds to the probability of selecting a specific node from the prefix as the next step. This decoding mechanism follows from the preceding design. With continuous embeddings and an instance-specific candidate set that is dynamically masked during construction, a fixed-vocabulary LM head is inapplicable. The output layer is therefore rebuilt around the prefix outputs themselves, which serve directly as the decoding keys, eliminating the need for a separate encoder.

Replacing Positional Encoding with Dynamic State Aggregator. In contrast to Rotary Position Embeddings (RoPE) (Su et al., 2023) employed in Zong et al. (2025), we adopt No Positional Encoding (NoPE). For CO problems, the historical partial trajectory primarily conveys node selection status rather than sequential order. For problems such as CVRP, critical contextual information (e.g., remaining vehicle capacity) is inherently tied to the most recently visited nodes. Rather than

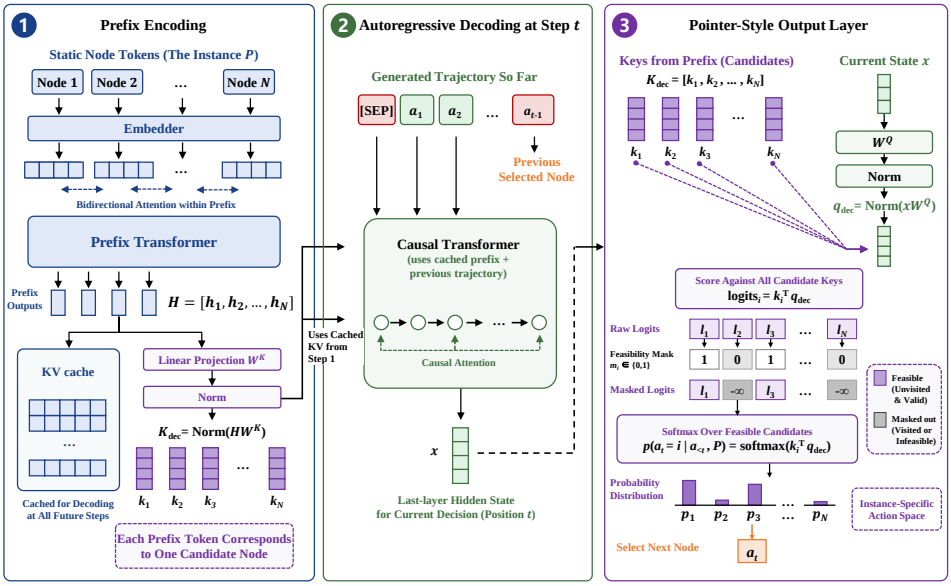

Figure 2: Overview of PrefixCO. The prefix phase employs bidirectional attention to process static features and initialize the KV cache and decoding keys. Initiated by the [SEP] token, the generation phase uses causal attention to query both the static prefix and the previous partial trajectory.

relying on positional encoding to infer this, we directly incorporate such information into the input embedding of the last node. This approach is more straightforward and easier for the model to learn.

3.2 Stabilizing RL with Non-Negative Reward Shaping

We empirically observed that directly applying the original POMO to training Prefix Transformers can encounter instability. To identify the underlying cause, we conducted a diagnostic experiment in which the policy was first pre-trained via supervised learning on optimal solutions, ensuring that the model started from a strong initial policy. We then switched to POMO-based RL with a highly conservative learning rate of 1×10^{-5}. As shown in Fig. 3, the results reveal a distinctive anomaly: while the *minimum* cost within each group remains stable, the *mean* cost gradually increases. Since the training loss continues to decrease smoothly, the optimization process appears normal under standard monitoring metrics, despite the fact that the policy is effectively degrading. We attribute this phenomenon to the high expressivity of Prefix Transformers, which allows to minimize the collective loss by aggressively suppressing the probability mass assigned to sub-optimal trajectories. Rather than shift the entire trajectory distribution toward better solutions, the model exploits the relative baseline by driving the probabilities of high-cost trajectories toward zero. Although this behavior reduces the training loss and preserves a high probability for the best-found solutions, it degrades the overall policy distribution, as reflected by the widening gap between the minimum and mean costs.

A remedy is to clamp the shaped rewards to be non-negative. We define the positive reward shaping:

$$\widehat{R}_{\text{pos.}}(\tau^i \mid \{\tau^j\}_{j=1}^{M}) = \max(0, R(\tau^i) - \frac{1}{M}\sum_{j=1}^{M} R(\tau^j)). \tag{5}$$

As illustrated in Fig. 3, this modification effectively resolves the aforementioned instability. Furthermore, inspired by the success of preference-based training (Pan et al., 2025a; Luo et al., 2025), we introduce a simple self-improvement reward shaping scheme:

$$\widehat{R}_{\text{si}}(\tau^i \mid \{\tau^j\}_{j=1}^{M}) = \mathbb{1}[R(\tau^i) = \max_{j\in[M]} R(\tau^j)], \tag{6}$$

which assigns 1 to the best trajectory in the group and 0 to all others. Note that this formulation remains strictly non-negative. Initial experiments show that this approach matches the performance of $\widehat{R}_{\text{pos.}}$, but it is significantly more efficient in terms of training resources: only the best trajectory

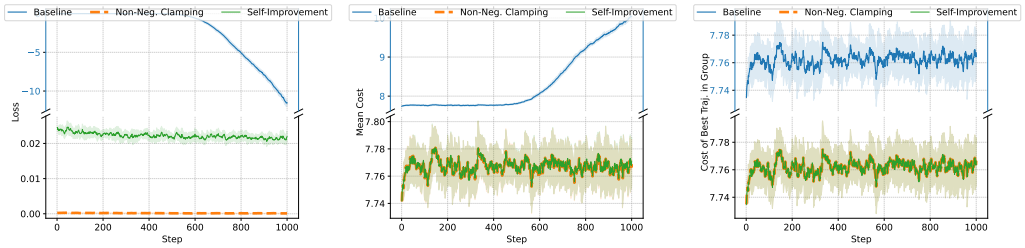

Figure 3: The shared initial policy is obtained via supervised learning on a dataset of optimal solutions, with a learning rate of 10^{-5} maintained during the following RL phase. Experimental results on TSP-100 reveal that while the policy trained using the POMO baseline undergoes performance degradation, the non-negative clamping and self-improvement reward reshaping strategies remain robust and yield nearly identical performance curves. Notably, under the POMO baseline, the cost of the best trajectory within each group remains stable, whereas the mean cost gradually increases, indicating a consistent loss in the learned policy caused by reward hacking.

requires gradient computation. In POMO-style training, where group sizes are typically 50 or 100, this reduces the majority of backward passes, leaving rollout as the dominant computational cost. Moreover, the substantial memory savings enable the use of larger batch sizes.

Although exclusively non-negative reward signals can theoretically hinder exploration, we observe no entropy collapse, and final performance remains robust. We hypothesize that this stems from the inherent complexity of CO combined with an effectively infinite set of training instances, preventing the model from memorizing a single decision pattern, thereby naturally preserving policy diversity. Tailored to the failure mode diagnosed above, this shaping provides precisely the stabilization that makes large-scale RL training of prefix transformers viable.

3.3 Additional Scaling Techniques

The techniques in this subsection complete the training recipe of PrefixCO. Each is summarized briefly here, with full descriptions deferred to Appendix B.

Grouped Query Attention (GQA). We incorporate GQA (Ainslie et al., 2023) to accelerate autoregressive decoding by sharing key and value heads across groups of query heads. Such acceleration is vital for RL, where massive rollout volumes make high-speed inference a practical necessity. In the CO domain, we observe no quality loss from this substitution (Table 5 in Appendix B).

Imitation warm-starting. Deep prefix transformers are difficult to train from scratch under RL (Zong et al., 2025). We therefore warm-start the policy with imitation learning on solutions produced by cheap local heuristics, using the 2-opt algorithm for TSP and local search (Ma et al., 2025b) for CVRP. The ablation in Fig. 5 in Appendix B demonstrates that this stage is essential for stable training.

4 Experiments

We compare PrefixCO with learning-based solvers on TSP and CVRP, with a focus on AR methods. We further evaluate scalability by benchmarking against non-AR solvers on TSP. For fair comparison with SL-trained baselines, we also report a supervised variant, PrefixCO-SL. All evaluations are conducted on a single GPU of NVIDIA A100, unless otherwise stated.

4.1 Experiments Settings

Problem Setup. Following standard generation procedures (Kool et al., 2018; Kwon et al., 2020; Drakulic et al., 2023; Luo et al., 2024; Pan et al., 2025a), we sample n nodes for TSP and n customer locations plus a depot for CVRP uniformly from the unit square $[0, 1]^2$. For CVRP, customer demands are drawn uniformly from the set $\{1, 2, \ldots, 9\}$, with the vehicle capacity is set to 50 for CVRP-100 and 100 for CVRP-500. The data pipeline varies by training paradigm: for RL, instances are generated on the fly without data reuse or replay, whereas for SL and IL, instances are pre-generated and labeled by expert solvers or weak heuristics. In the TSP setting, the starting node is sampled uniformly

Table 1: Comparison with non-autoregressive solvers on TSP under supervised learning settings.

Method	TSP-100 (1280 inst.)			TSP-500 (128 inst.)			TSP-1000 (128 inst.)		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Concorde	7.76	0.000%	5.1m	16.55	0.000%	39.8m	23.12	0.000%	3.0h
DIFUSCO (T_s=50) G+2Opt	7.78	0.28%	18.9m	16.80	1.50%	4.7m	23.55	1.89%	14.4m
DIFUSCO (T_s=100) S+2Opt	7.76	0.06%	30m	16.69	0.87%	19.1m	23.42	1.31%	51.9m
T2T (T_s=50,T_g=30) G+2Opt	7.76	0.07%	28.6m	16.68	0.82%	6.5m	23.44	1.40%	19.7m
T2T (T_s=50,T_g=30) S+2Opt	7.76	0.02%	45.1m	16.63	0.48%	19.7m	23.37	1.07%	51.1m
Fast T2T (T_s=5,T_g=5) G+2Opt	7.76	0.03%	4.2m	16.61	0.39%	2.2m	23.25	0.58%	8.6m
Fast T2T (T_s=5,T_g=5) S	–	–	–	16.72	1.02%	7.2m	23.68	2.44%	19.0m
Fast T2T (T_s=5,T_g=5) S+2Opt	7.76	0.01%	8.3m	16.58	0.21%	6.9m	23.22	0.42%	18.3m
GenSCO (C=40)	–	–	–	16.55	0.016%	18s	23.18	0.258%	46s
GenSCO (C=40) 2Opt	–	–	–	16.55	0.012%	18s	23.13	0.046%	58s
GenSCO (C=160) 2Opt	–	–	–	16.55	0.010%	1.2m	23.13	0.036%	3.9m
PrefixCO-SL (aug×8, G=1)	7.756	0.0002%	4s	16.547	0.0080%	5s	23.120	0.0076%	14s
PrefixCO-SL (aug×8, G=8)	7.756	0.0000%	0.38m	16.546	0.0033%	0.46m	23.119	0.0045%	1.5m
PrefixCO-SL (aug×8, G=32)	–	–	–	16.546	0.0028%	1.7m	23.119	0.0037%	5.8m
PrefixCO-SL (aug×8, G=128)	–	–	–	16.546	0.0023%	6.5m	23.119	0.0031%	23.1m
PrefixCO-SL (aug×8, S=1)	7.756	0.0003%	4s	16.549	0.0209%	5s	23.121	0.0121%	14s
PrefixCO-SL (aug×8, S=8)	7.756	0.0000%	0.38m	16.547	0.0053%	0.46m	23.119	0.0050%	1.5m
PrefixCO-SL (aug×8, S=32)	–	–	–	16.546	0.0023%	1.7m	23.119	0.0030%	5.8m
PrefixCO-SL (aug×8, S=128)	–	–	–	16.546	0.0009%	6.5m	23.118	0.0016%	23.1m
PrefixCO-SL (aug×8, S=512)	–	–	–	16.546	0.0004%	26.0m	23.118	0.0010%	92.3m

across all paradigms; for SL/IL, this is implemented by cyclically shifting the expert solution until the selected node is at the sequence's beginning. In contrast, for simplicity, the CVRP start node is fixed to the depot, as we adopt a solution format similar to that used in Ma et al. (2021). During CVRP solution generation, consecutive depots are prohibited in the interior of the route through masking, except when no non-depot node remains selectable.

Decoding Settings. For decoding, we adopt a multi-trajectory strategy, the standard convention for autoregressive solvers, noting that PrefixCO remains inherently compatible with advanced search mechanisms such as beam search (Drakulic et al., 2023) or the reconstruction-based sampling procedure RRC of LEHD (Luo et al., 2024). The start node selection policy mirrors the procedure used during training. Specifically, for TSP, multiple trajectories are generated using either greedy or sampling-based decoding. For CVRP, however, the fixed depot starting location necessitates the exclusive use of sampling to ensure trajectory diversity and prevent the generation of redundant solutions. We use S and G to denote the number of sampling trajectories and the number of greedy trajectories, respectively. Furthermore, we consistently incorporate the instance augmentation technique introduced in POMO (Kwon et al., 2020) across all evaluations, which is indicated as "aug×8" in the tables.

4.2 Main Results

TSP Results. Table 2 presents the performance comparison with other autoregressive solvers. Our PrefixCO achieves near-optimal solution quality on TSP-100 (0.0001%) within 5.8m, and reduces the Gap by **30** times on TSP-500, compared with previous SOTA autoregressive baselines. With a minimal decoding budget (aug×8, S=1), PrefixCO significantly outperforms all other autoregressive methods while maintaining highly competitive inference times (**4s** for TSP-100 and **5s** for TSP-500). When the budget is slightly increased (S=8), our model achieves a Gap of 0.0003% in just 0.38 minutes on TSP-100. Beyond efficiency, PrefixCO addresses the **scalability** limitations inherent in prior RL-based solvers. Many such methods either experience a significant collapse in solution quality when evaluating on $N = 500$ or struggle with the complexities of end-to-end training. In contrast, PrefixCO maintains high precision across scales, achieving an impressive 0.0077% gap on TSP-500, outperforming SOTA baselines by a substantial margin. Furthermore, PrefixCO exhibits exceptional computational efficiency. To generate 8 trajectories (aug×8, S=1) on TSP-500, our model requires only 5 seconds. In contrast, heavy-decoder solvers such as BQ-NCO and LEHD consume 1.8 minutes and 8 seconds, respectively, to generate just a single trajectory. This highlights that despite having a theoretically heavier decoder, PrefixCO is highly optimized for high-throughput inference, achieving a significantly better performance-to-latency ratio than existing heavy-decoder baselines.

Table 2: Comparison with AR solvers on TSP and CVRP.

Method	TSP-100 (1280 inst.)			TSP-500 (128 inst.)			CVRP-100 (1280 inst.)			CVRP-500 (100 inst.)		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Concorde	7.756	0.000%	5.1m*	16.55	0.000%	39.8m*	–	–	–	–	–	–
LKH-3	7.756	0.000%	14.1m*	16.55	0.000%	15.0m*	–	–	–	–	–	–
HGS	–	–	–	–	–	–	15.550	0.000%	13h	37.154	0.000%	8.3h*
SL-based Autoregressive Solvers												
BQ-NCO Greedy	7.782	0.343%	9s	16.754	1.256%	0.68m	–	–	–	–	–	–
BQ-NCO BS16	7.757	0.016%	1.7m	16.637	0.551%	1.8m	15.794	1.572%	2.7m	38.232	2.899%	4.7m
LEHD Greedy	7.799	0.5502%	2s	16.802	1.547%	8s	–	–	–	–	–	–
LEHD RRC 100	7.757	0.0101%	0.90m	16.605	0.355%	3.8m	15.676	0.814%	1.1m	38.070	2.465%	3.1m
LEHD RRC 1000	7.756	0.0019%	7.9m	16.575	0.175%	37.5m	15.617	0.433%	9.5m	37.734	1.561%	33.0m
RL-based Autoregressive Solvers												
POMO aug×8	7.77	0.13%	8s*	20.22	22.19%	1.0m*	15.750	1.287%	8s	–	–	–
POMO+EAS AS+G	7.76	0.05%	38.4m*	24.54	48.22%	11.6h*	–	–	–	–	–	–
POMO+PO aug×8	7.759	0.0425%	8s	–	–	–	15.733	1.181%	0.17m	–	–	–
ReLD aug×8	–	–	–	–	–	–	15.785	1.509%	7s	38.974	4.898%	8s
PrefixCO (aug×8, G=1)	7.756	0.0028%	4s	16.550	0.0246%	5s	–	–	–	–	–	–
PrefixCO (aug×8, G=8)	7.756	0.0005%	0.38m	16.548	0.0135%	0.46m	–	–	–	–	–	–
PrefixCO (aug×8, G=32)	7.756	0.0002%	1.5m	16.547	0.0099%	1.7m	–	–	–	–	–	–
PrefixCO (aug×8, G=128)	7.756	0.0001%	5.8m	16.547	0.0077%	6.5m	–	–	–	–	–	–
PrefixCO (aug×8, S=1)	7.756	0.0030%	4s	16.551	0.0287%	5s	15.664	0.734%	5s	37.646	1.322%	8s
PrefixCO (aug×8, S=8)	7.756	0.0003%	0.38m	16.548	0.0131%	0.46m	15.611	0.391%	0.5m	37.553	1.072%	0.4m
PrefixCO (aug×8, S=32)	7.756	0.0002%	1.5m	16.547	0.0093%	1.7m	15.593	0.280%	1.7m	37.510	0.956%	1.3m
PrefixCO (aug×8, S=128)	7.756	0.0001%	5.8m	16.547	0.0064%	6.5m	15.582	0.205%	6.8m	37.463	0.830%	5.0m
PrefixCO (aug×8, S=512)	7.756	0.0001%	23.0m	16.547	0.0056%	26.0m	15.574	0.157%	27.1m	37.430	0.742%	19.6m

CVRP Results. Table 2 clear advantages over autoregressive baselines. For CVRP-100, with a modest budget of (aug×8, S=8), PrefixCO surpasses all autoregressive baselines on solution quality, which only takes **0.5m**. On larger CVRP-500, some RL-based autoregressive solvers become infeasible. In contrast, PrefixCO (S=1) already outperforms all autoregressive baselines, delivering a **1.322%** gap in just **8s**. This is significantly more efficient than LEHD RRC 1000, which takes 33m to reach a 1.561% gap. Furthermore, as the sampling budget increases to S=512, PrefixCO scales effectively, narrowing the gap to **0.157%** for CVRP-100 and **0.742%** for CVRP-500. This represents a **66%** and **50%** reduction in the gap, respectively, showcasing the model's superior scalability and its highly competitive trade-off between solution quality and inference latency.

Comparison with Non-Autoregressive Solvers on TSP. In Table 1, we compare our PrefixCO with other non-autoregressive solvers, including DIFUSCO Sun & Yang (2023), T2T Li et al. (2023), Fast T2T Li et al. (2024), and GenSCO Li et al. (2025a), on large-scale TSP-100, TSP-500, and TSP-1000. On TSP-100, PrefixCO-SL reaches the optimal solution within **0.38m**. On TSP-1000, PrefixCO-SL (aug×8, G=8) delivers an **87%** gap reduction and is **2.6x** faster than the SOTA GenSCO (C=160, 2-opt); this reduction improves to **97%** with a larger inference budget (S=512). Notably, PrefixCO-SL achieves high performance without relying on post-processing techniques like 2-opt. In contrast, non-autoregressive baselines like Fast T2T and GenSCO degrade significantly when 2-opt is disabled, highlighting the superior learning capability of our approach.

4.3 Ablation Studies and Further Investigations

Impact of Non-Negative Reward Shaping. Fig. 3 shows that reward hacking phenomenon is effectively mitigated, resulting in stabilized training. Tables 2 proves that with self-improvement reward shaping, model can still benefit from the diversity-based multi-trajectory decoding strategy, confirming successful diversity preserving.

Continuous Embedder versus Discrete Tokenizer. We conduct an ablation against a discrete tokenizer on the same backbone. Following the tokenizer design in UniCO (Zong et al., 2025), where node features are flattened and each scalar is individually discretized into tokens, we adapt this scheme to our architecture: since we do not flatten node features, we tokenize each scalar separately and then concatenate the resulting embeddings. All other settings remain exactly the same as UniCO, including the hyperparameters of the discrete tokenizer. For rapid experimentation, we adopt the supervised training setup, reduce the network depth from 64 to 16 layers, and cut the training steps to one third of the original. We compare both the final solving performance and the training loss, which directly reflect the representational capacity of each method. As shown in Table 3, across all settings the continuous embedder consistently and substantially outperforms the discrete tokenizer. More importantly, the performance gap widens noticeably as the problem size increases. This can be

Table 3: Continuous embedder versus discrete tokenizer on the same backbone. Δ denotes performance variation.

	TSP-100		TSP-500	
	Discrete	Continuous	Discrete	Continuous
Gap (aug×8, S=32)	0.2076%	0.0000%	6.0322%	0.0239%
Training loss	0.1796	0.0345	0.3182	0.0505
Δ Gap	−0.2076%		−6.0083%	
Δ Training loss	−0.1451		−0.2677	

Table 4: Inference time of PrefixCO on TSP-100 (aug×8, S=1) with and without KV caching.

	Inference time
PrefixCO w/o KV cache	31m
PrefixCO w/ KV cache	4s

explained by the fact that, with a fixed coordinate range, larger instances place points more densely, which amplifies the adverse effects of rounding errors inherent in the discrete tokenization process.

Ablations of the supporting techniques. The studies for grouped-query attention and imitation warm-starting, as well as those for the exploratory extension, are reported in Appendix B alongside the descriptions of the techniques themselves.

Prefix Design Enables Efficient KV-Cached Decoding. As shown in Table 4, on TSP-100 (aug×8, S=1) inference without the KV cache takes approximately 31 minutes, whereas the same model with the cache finishes in just 4 seconds, a speedup of nearly 465×. This is a substantial improvement, and it is what makes large-scale inference of an expressive autoregressive solver affordable: the second-scale timings reported for PrefixCO in Table 2 are attainable only with cached decoding.

The decisive factor for this gain is the prefix-autoregressive design, making KV-cached decoding possible for heavy-decoder models. Light-decoder baselines such as POMO can reuse static node embeddings only because their decoder consists of a single layer, while expressive multi-layer decoders e.g. BQ-NCO and LEHD cannot cache at all. The prefix design removes this trade-off: caching stays valid while the decoder retains full multi-layer capacity over the evolving trajectory.

4.4 Extension to Maximum Independent Set

To examine whether the prefix-autoregressive design extends beyond routing, we adapt PrefixCO to the Maximum Independent Set (MIS) problem, whose solution is a node subset rather than a sequence. Adapting PrefixCO requires only a task-specific representation of graph: node features are encoded as prefix tokens, while graph edges are incorporated as additive attention biases. The model then autoregressively selects feasible nodes, with already selected nodes and their neighbors masked from subsequent decisions. Following the standard evaluation protocol for MIS (Sun & Yang, 2023), we generate RB graphs with 200 to 300 nodes and report the objective, gap, and inference time in Table 4.

Figure 4: Results on MIS with RB graphs of 200-300 nodes. Each entry reports the objective, the gap, and the inference time. All methods are evaluated on a single NVIDIA RTX 4090 under the same implementation environment.

Method	Obj.	Gap	Time
DIFUSCO (T_s=100, S) (Sun & Yang, 2023)	19.13	4.79%	20.5m
T2T (T_s=5, T_g=5, S) (Li et al., 2023)	19.38	3.53%	30.3m
Fast T2T (T_s=5, T_g=5, S) (Li et al., 2024)	19.70	1.90%	7.0m
GenSCO (C=1K) (Li et al., 2025b)	19.91	0.96%	2.2m
GenSCO (C=2K) (Li et al., 2025b)	19.97	0.64%	4.4m
GenSCO (C=4K) (Li et al., 2025b)	20.01	0.43%	8.8m
GenSCO (C=8K) (Li et al., 2025b)	20.04	0.29%	17.6m
GenSCO (C=16K) (Li et al., 2025b)	20.06	0.22%	35.2m
PrefixCO-SL (S=12800)	20.05	0.23%	2.4m
PrefixCO-SL (S=25600)	20.07	0.17%	4.7m
PrefixCO-SL (S=51200)	20.07	0.14%	9.5m
PrefixCO-SL (S=102400)	20.08	0.12%	19.0m

PrefixCO achieves a 0.23% gap in 2.4 minutes and improves to a 0.12% gap with a larger sampling budget, matching or surpassing the strongest non-autoregressive baselines at comparable or lower inference time. This result is particularly encouraging because MIS benchmarks have largely been dominated by non-autoregressive methods. We present it as evidence that the architectural principle can extend to a non-routing graph problem, while the primary scope of this paper remains routing-style NCO. A broader investigation across different CO output structures remains future work.

5 Conclusion

This paper has presented PrefixCO, an effective and efficient autoregressive paradigm for routing-style NCO that presents a harmonious balance between expressiveness and computational scalability. By reimagining the architectural foundation with dynamic embeddings and a prefix-based design, alongside a stabilized RL scheme incorporating imitation learning stage and non-negative reward shaping, we address the critical bottlenecks previously associated with autoregressive solvers.

AI USE STATEMENT

AI tools were used to aid or polish writing. Specifically, they were used for language editing, grammar and spelling checking, improving clarity and readability, and refining the academic style of the manuscript. All AI-generated suggestions were reviewed, verified, and revised by the authors, who take full responsibility for the final content.

AI tools were also used to draft sections of the paper. Specifically, AI assistance was used to prepare initial outlines for the Methodology part directly from our code; these outlines were subsequently checked, edited, and rewritten by the authors.

ETHICS STATEMENT

This research was conducted in accordance with the ICLR Code of Ethics and with a strong commitment to responsible research practices. It does not involve human subjects, sensitive personal data, or potentially harmful applications. We have sought to maintain transparency, reproducibility, and scientific integrity throughout the study, including by reporting methods and results accurately. We have no conflicts of interest involving funding sources or affiliations that could influence this work. Our aim is for this research to contribute positively to the machine learning community and to society more broadly by promoting fairness, accessibility, and open scientific inquiry. If unforeseen broader impacts arise, we remain committed to addressing them responsibly.

REPRODUCIBILITY STATEMENT

We are committed to upholding high standards of scientific excellence and transparency in accordance with the ICLR Code of Ethics. To ensure the reproducibility of our results, we have documented the full methodology in the main text and appendices. Sec. 3 specifies the prefix-transformer architecture (Subsec. 3.1), the non-negative reward shaping that stabilizes RL training (Subsec. 3.2), and the complementary scaling techniques of grouped query attention and imitation warm-starting (Subsec. 3.3), with complete descriptions of these techniques and of the multi-token prediction heads deferred to Appendix B. The instance generation procedures and the decoding protocol, including the greedy and sampling trajectory budgets and the eight-fold instance augmentation, are stated in the Experiments Settings part of Sec. 4. Model and training configurations, hyperparameters, hardware, dataset construction, and the solvers used to label the test sets are reported in Appendix C. The autoregressive baselines are reproduced from their publicly available code, and the meaning of every decoding parameter in Tables 1 and 2 is fixed in Appendix D.1, so that each reported comparison is fully specified. Appendix D additionally reports parameter counts (Subsec. D.5), the controlled comparisons under matched hardware, trajectory, and wall-clock budgets (Subsec. D.6), and further diagnostics for reward shaping and imitation warm-starting (Subsec. D.7), while Sec. 4.4 details the maximum independent set extension. Source code and evaluation scripts will be made publicly available upon acceptance.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to handle complex constraints for vehicle routing problems, 2024. URL https://arxiv.org/abs/2410.21066.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin

Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023. URL https://arxiv.org/abs/2212.06817.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. URL https://arxiv.org/abs/2106.01345.

Lvda Chen, Yang Li, and Junchi Yan. Maskco: Masked generation drives effective representation learning and exploiting for combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.

Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems*, 35:26350–26362, 2022.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. Goal: A generalist combinatorial optimization agent learner, 2025. URL https://arxiv.org/abs/2406.15079.

Zhanhong Fang, Debing Wang, Jinbiao Chen, Jiahai Wang, and Zizhen Zhang. UCPO: A universal constrained combinatorial optimization method via preference optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2511.10148.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7474–7482, 2021.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction, 2024. URL https://arxiv.org/abs/2404.19737.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pp. 24–50, 2017.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems, 2025. URL https://arxiv.org/abs/2503.00753.

Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2022.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem, 2021. URL https://arxiv.org/abs/2106.02039.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in neural information processing systems*, 2021.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *arXiv preprint arXiv:2205.13209*, 2022.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34:5138–5149, 2021.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yang Li, Lvda Chen, haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2025a.

Yang Li, Lvda Chen, Haonan Wang, Runzhong Wang, and Junchi Yan. Generation as search operator for test-time scaling of diffusion-based combinatorial optimization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=9JM03CQwzC.

Yang Li, Jiale Ma, Wenzheng Pan, Runzhong Wang, Haoyu Geng, Nianzu Yang, and Junchi Yan. Unify ml4tsp: Drawing methodological principles for tsp and beyond from streamlined design space of learning and search. In *The Thirteenth International Conference on Learning Representations*, 2025c.

Zijun Liao, Jinbiao Chen, Debing Wang, Zizhen Zhang, and Jiahai Wang. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. In *Forty-second International Conference on Machine Learning*, 2025.

Attila Lischka, Jiaming Wu, Morteza Haghir Chehreghani, and Balázs Kulcsár. A great architecture for edge-based graph problems like tsp, 2024. URL https://arxiv.org/abs/2408.16717.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization, 2024. URL https://arxiv.org/abs/2402.16891.

Ruiwu Liu, Xiaocen Li, and Kit S Lam. Combinatorial chemistry in drug discovery. *Current Opinion in Chemical Biology*, 38:117–126, 2017. ISSN 1367-5931. doi: https://doi.org/10.1016/j.cbpa.2017.03.017. URL https://www.sciencedirect.com/science/article/pii/S1367593117300534. Next Generation Therapeutics.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36, 2024.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=TbTJJNjumY.

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. COExpander: Adaptive solution expansion for combinatorial optimization. In *Forty-second International Conference on Machine Learning*, 2025a.

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Ml4co-bench-101: Benchmark machine learning for classic combinatorial problems on graphs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025b. URL https://openreview.net/forum?id=ye4ntB1Kzi.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 11096–11107. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/5c53292c032b6cb8510041c54274e65f-Paper.pdf.

Dian Meng, Zhiguang Cao, Jie Gao, Yaoxin Wu, and Yaqing Hou. Uniteformer: Unifying node and edge modalities in transformers for vehicle routing problems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=BRklmFlCsD.

Dian Meng, Zhiguang Cao, Yaoxin Wu, Yaqing Hou, Hongwei Ge, and Qiang Zhang. Eformer: An effective edge-based transformer for vehicle routing problems. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 8582–8590. International Joint Conferences on Artificial Intelligence Organization, 8 2025b. doi: 10.24963/ijcai.2025/954. URL https://doi.org/10.24963/ijcai.2025/954. Main Track.

Yimeng Min, Yiwei Bai, and Carla P. Gomes. Unsupervised learning for solving the travelling salesman problem. In *Advances in Neural Information Processing Systems*, 2023.

Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems, 2025a. URL https://arxiv.org/abs/2505.08735.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=yEwakMNIex.

Wenzheng Pan, Hao Xiong, Jiale Ma, Wentao Zhao, Yang Li, and Junchi Yan. UniCO: On unified combinatorial optimization via problem reduction to matrix-encoded general TSP. In *The Thirteenth International Conference on Learning Representations*, 2025c.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *arXiv preprint arXiv:2210.04123*, 2022.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022. URL https://arxiv.org/abs/2205.06175.

Sebastian Sanokowski, Wilhelm Berghammer, Sepp Hochreiter, and Sebastian Lehner. Variational annealing on graphs for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:63907–63930, 2023.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Forty-first International Conference on Machine Learning*, 2024a.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization, 2024b. URL https://arxiv.org/abs/2406.01661.

Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022. URL https://arxiv.org/abs/2206.11251.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JV8Ff0lgVV.

Matthew Veres and Medhat Moussa. Deep learning for intelligent transportation systems: A survey of emerging trends. *IEEE Transactions on Intelligent Transportation Systems*, 21(8):3152–3168, 2020. doi: 10.1109/TITS.2019.2929020.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60 (3):611–624, 2012.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Haoyu Peter Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Hang Yi, Ziwei Huang, Zhiguang Cao, and Yining Ma. Radar: Learning to route with asymmetry-aware distance representations. In *International Conference on Learning Representations (ICLR)*, 2026.

Biao Zhang and Rico Sennrich. Root mean square layer normalization, 2019. URL https://arxiv.org/abs/1910.07467.

Changliang Zhou, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization. *CoRR*, 2024a.

Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Chi Xu. Mvmoe: Multi-task vehicle routing solver with mixture-of-experts, 2024b. URL https://arxiv.org/abs/2405.01029.

Zefang Zong, Xiaochen Wei, Guozhen Zhang, Chen Gao, Huandong Wang, and Yong Li. Solving diverse combinatorial optimization problems with a unified model, 2025. URL https://openreview.net/forum?id=Kc3yoIL5oR.

A Related Works

A.1 Neural Combinatorial Optimization

Autoregressive (AR) Methods. AR solvers cast CO as a Markov decision process, where a neural policy constructs solutions by sequentially selecting actions. **1) RL-based AR Solvers.** RL is the dominant training paradigm for mainstream AR solvers (Kool et al., 2018; Kim et al., 2022; Kwon et al., 2020; 2021; Kim et al., 2021; Zhou et al., 2024a; Huang et al., 2025), which typically employ lightweight single-layer decoders to maximize rollout efficiency. Recent advances mainly improve training, e.g., preference-based optimization (Fang et al., 2026; Liao et al., 2025; Pan et al., 2025a) for better sample efficiency, or strengthen encoders to better exploit edge information (Meng et al., 2025b; Lischka et al., 2024; Meng et al., 2025a). More broadly, RL-based AR solvers have been extended to complex-constrained CO variants, particularly in multi-task learning and adaptation settings (Pan et al., 2025c; Zhou et al., 2024b; Liu et al., 2024; Yi et al., 2026). Nevertheless, their overly simplified decoders remain an architectural bottleneck: limited capacity to model context-dependent patterns restricts expressiveness and impedes scalability, even beyond 100-node routing instances. **2) SL-based AR Solvers.** A smaller body of work adopts supervised learning (SL) in the AR regime, training models to imitate expert trajectories on small-scale instances, e.g., size 100, and generalize to larger ones (Drakulic et al., 2023; Luo et al., 2024; Drakulic et al., 2025). To capture dynamic decision patterns, these methods often rely on heavy multi-layer decoders, but suffer from low training efficiency. Since they usually supervise only one token per trajectory at each step, they underutilize the parallel training capability of Transformer architectures.

Non-autoregressive Methods. NAR solvers instead predict likelihoods (e.g. N^2 entries for edges in TSP) in a global, one-shot manner, which are then decoded into feasible solutions. Early methods adopt supervised graph convolutional networks (Hudson et al., 2022; Joshi et al., 2019; Fu et al., 2021), and recent work enhances distribution modeling via discrete diffusion (Sun & Yang, 2023; Li et al., 2023; 2024; Chen et al., 2026; Ma et al., 2025a; Li et al., 2025c; Ma et al., 2025b). A smaller line explores unsupervised heatmap learning (Qiu et al., 2022; Min et al., 2023; Wang & Li, 2023; Sanokowski et al., 2023; 2024a); however, they struggle even on relatively simple TSP, since routing objectives are difficult to optimize with naive penalty-based losses. In general, though capable of capturing solution distributions at scale, NAR approaches often require massive, off-the-shelf (near-)optimal labels, which are costly to obtain, and their global predictions offer less flexibility than AR methods in handling complex constraints, thereby limiting practical applicability.

A.2 Next Token Prediction in Decision Making with Transformers

Next-token prediction has become a dominant paradigm for decision making, pioneered by Decision Transformer (Chen et al., 2021) and further extended by Trajectory Transformer (Janner et al., 2021), which recasts RL as conditional sequence modeling and leverages transformer capacity for stability, long-horizon reasoning, and scalability. Beyond RL, imitation learning has also shown effectiveness under this paradigm (Reed et al., 2022; Shafiullah et al., 2022; Brohan et al., 2023). In NCO, solution construction is naturally formulated as sequential decision making, where each step predicts the next action token conditioned on the partial solution (Kool et al., 2018; Kim et al., 2022; Kwon et al., 2020; 2021; Kim et al., 2021; Zhou et al., 2024a; Huang et al., 2025; Drakulic et al., 2023; Luo et al., 2024; Drakulic et al., 2025).

B Details of Additional Scaling Techniques

This appendix details the additional scaling techniques summarized in Sec. 3.3, and additionally documents the exploratory extension based on multi-token prediction (MTP), for completeness and reproducibility.

B.1 Grouped Query Attention

To further improve decoding efficiency, we incorporate Grouped Query Attention (GQA) (Ainslie et al., 2023), which accelerates the autoregressive decoding of large-scale models by sharing key and value heads across groups of query heads. In natural language tasks, this substitution typically trades

model expressiveness for computational speed, whereas our experiments indicate no such loss within the CO domain: decoding is significantly accelerated without compromising solution quality. This efficiency is essential in RL, whose massive rollouts demand fast inference in practice.

As shown in Table 5, the integration of GQA significantly accelerates decoding compared to standard Multi-Head Attention (MHA), while the optimality gap remains identical or even slightly improves. This gain is likely attributable to enhanced training dynamics: across all training phases for TSP-100, TSP-500, and TSP-1000, the training loss of the GQA-equipped model consistently matched or was lower than that of the MHA baseline. GQA therefore yields substantial decoding efficiency without any discernible loss in model expressiveness.

Table 5: Ablation of GQA on TSP-100. GQA-1: GQA with 1 group, MHA: Multi-head Attention.

	GQA-1	MHA
PrefixCO-SL (aug×8, S=1)	**0.0003%, 4s**	0.0005%, 14s
PrefixCO-SL (aug×8, S=8)	**0.0000%, 0.38m**	**0.0000%**, 1.8m

B.2 Imitation Learning as Policy Initialization

Prefix Transformers are notoriously difficult to train for CO, even in supervised learning settings (Zong et al., 2025), and this difficulty is further magnified under RL, where the initial policy serves a dual purpose: providing a sound parameter initialization and enabling effective exploration to gather meaningful reward signals. To address this, we warm-start the model with Imitation Learning (IL), a technique consistently shown to improve the convergence of RL agents. We use the 2-opt algorithm as the reference policy for TSP and local search (Ma et al., 2025b) for CVRP. Further diagnostics on the necessity and low cost of this warm-start stage are provided in Appendix D.7.

Fig. 5 shows that incorporating an IL stage prior to RL significantly improves training stability. The model trained without the IL stage fails to achieve a reasonable optimality gap by the end of training.

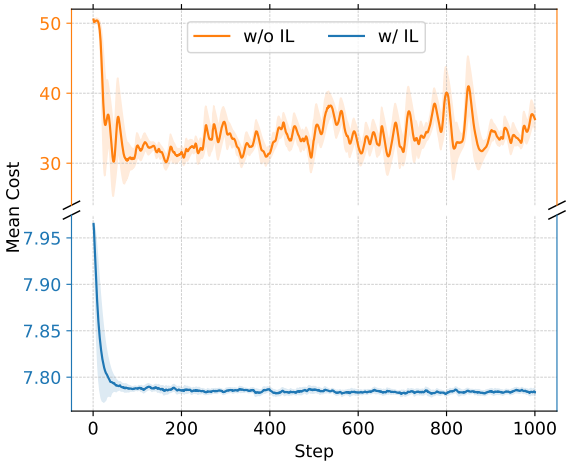

Figure 5: Comparison of PrefixCO trained with and without imitation learning on TSP-100.

B.3 Multi-Token Prediction

Inspired by recent progress in natural language processing, where multi-token prediction (MTP) improves both model performance and inference efficiency through speculative decoding (Gloeckle et al., 2024; DeepSeek-AI et al., 2025), we adapt this paradigm to CO. Specifically, to enable the simultaneous prediction of the next m tokens, we introduce m independent query projections, denoted as $W_1^Q, W_2^Q, \ldots, W_m^Q \in \mathbb{R}^{d\times d}$. The probability distribution p_i for the i-th future token is computed via:

$$p_i = \text{softmax}(\text{Norm}(HW^K) \cdot \text{Norm}(xW_i^Q)^\top), i \in [m]. \tag{7}$$

The handling of task-specific constraints differs between the training and inference phases. During training, we deliberately omit validity masks, which serves two purposes: it encourages the model to implicitly learn the underlying task-specific constraints from the data, and it prevents information leakage that could arise if future token constraints were revealed through the mask. During inference with self-speculative decoding (i.e., generating multiple candidate tokens to accelerate the autoregressive process), we apply validity masks sequentially: the mask for the i-th node is constructed only after the $(i-1)$-th node in the sequence has been determined, ensuring that all proposed candidates satisfy the feasibility requirements of the problem.

While MTP integrates seamlessly with SL, its application to RL is more nuanced due to a training-inference discrepancy: during training, every token in a sequence is supervised by all m decoding query heads, whereas during inference a token is typically generated by only one head. Despite this potential inconsistency, recent work such as DeepSeek-R1 (Guo et al., 2025) has applied MTP to RL rollouts with significant performance gains, suggesting that the shared representation learning encouraged by MTP may outweigh the discrepancy in certain contexts. We examine the empirical effectiveness of MTP for CO tasks in this subsection and discuss where it helps or degrades performance in Appendix D.8.

Impact of MTP for SL. As shown in Table 6, introducing MTP for supervised training consistently enhances the solution quality of PrefixCO-SL across different sampling budgets. Specifically, with a sampling size of S=8, the inclusion of MTP-4 reduces the optimality gap from 0.0071% to 0.0053%. This performance boost persists when the search budget increases. These results suggest that the MTP objective encourages the model to capture more complex dependencies and internalize a more global "planning" perspective during the supervised learning process, leading to more precise decision-making during inference.

Impact of MTP for RL. We use MTP only for gradient computation during RL and disable self-speculative decoding in rollouts. As shown in Table 6, MTP still improves RL training on TSP, reducing the optimality gap of PrefixCO (aug×8, S=1) by about 33%. However, the RL block on CVRP-100 in the same table shows a slight degradation. This difference may stem from task-dependent capacity requirements: prior work suggests that the benefit of MTP can depend on whether model capacity exceeds a certain threshold (Gloeckle et al., 2024). The additional constraints in CVRP may raise this threshold, making the current model insufficient to fully leverage MTP.

Results of Self-Speculative Decoding. We apply self-speculative decoding (SSD) to a model trained with MTP. To minimize complexity with variable-length sequences and avoid dependencies on specialized inference frameworks such as vLLM (Kwon et al., 2023), we currently do not use a verification mechanism. We observe distinct behavioral differences between greedy and sampling decoding strategies. Table 7 shows that applying SSD-2 (generating two tokens per forward pass) consistently reduces inference time by approximately 35-40% across all settings. We notice that the impact on solution quality varies significantly between decoding methods. When using greedy decoding, the performance degradation caused by SSD is small or negligible. In contrast, sampling scenarios exhibit a more noticeable drop in performance when SSD is enabled. These results show that SSD can substantially accelerate inference, but also highlight the importance of calibration when speculative tokens are used for sampling. Future work may further close this gap by incorporating temperature tuning or a lightweight verification mechanism.

C Experiment Details

C.1 Autoregressive Baseline Details

To ensure a fair comparison among autoregressive solvers, we always adopt the data augmentation technique introduced in Kwon et al. (2020) for all autoregressive baselines and our PrefixCO, which has become a standard practice in the field. Results for these baselines are reproduced using their publicly available code. For autoregressive solvers with lightweight decoders, the provided implementations are already well-engineered and include integrated KV-cache optimization for inference (as evidenced by the "set_kv" method in the POMO-series codebase, ReLD, and others).

Table 6: Impact of MTP.

Setting	w/o MTP	MTP-4
SL on TSP-500		
PrefixCO-SL (aug×8, S=8)	0.0071%	**0.0053%**
PrefixCO-SL (aug×8, S=32)	0.0028%	**0.0023%**
RL on TSP-100		
PrefixCO (aug×8, S=1)	0.0045%	**0.0030%**
PrefixCO (aug×8, S=8)	0.0004%	**0.0003%**
RL on CVRP-100		
PrefixCO (aug×8, S=8)	**0.391%**	0.441%
PrefixCO (aug×8, S=32)	**0.280%**	0.326%

Table 7: SSD results on TSP-500. SSD-2 generates two tokens per forward pass.

	w/o SSD	SSD-2
PrefixCO-SL (aug×8, G=8)	**0.0033%**, 0.5m	0.0042%, **0.3m**
PrefixCO-SL (aug×8, G=32)	**0.0028%**, 1.7m	0.0031%, **1.1m**
PrefixCO-SL (aug×8, G=128)	**0.0023%**, 6.5m	**0.0023%**, **4.1m**
PrefixCO-SL (aug×8, S=8)	**0.0053%**, 0.5m	0.0224%, **0.3m**
PrefixCO-SL (aug×8, S=32)	**0.0023%**, 1.7m	0.0094%, **1.1m**
PrefixCO-SL (aug×8, S=128)	**0.0009%**, 6.5m	0.0036%, **4.1m**

Note that, by default, the number of trajectories these light-decoder solvers rollout during inference equals the number of nodes in the problem instance.

C.2 Hardware Resources

All test evaluations are performed on a single GPU of the NVIDIA A100, unless otherwise specified.

C.3 Datasets

TSP test sets are labeled using the Concorde exact solver (Applegate et al., 2006), while CVRP test sets are labeled using HGS (Vidal et al., 2012).

For supervised learning on TSP, LKH-3 (Helsgaun, 2017) is employed to collect a high-quality training dataset with near-optimal reference solutions. The dataset contains 128,000 instances, consistent with (Li et al., 2024; 2025a; Chen et al., 2026).

C.4 Model and Training Configurations.

For TSP, we employ a PrefixCO decoder with 256-dimensional embeddings and 64 layers, trained with a peak learning rate of 1e-3 under a cosine annealing schedule. For the CVRP, we use a configuration with 512-dimensional embeddings and 32 layers, with a reduced peak learning rate of 5e-4 for stability. In both settings, the attention head dimension is fixed at 32, and GQA utilizes a single group. For SL training on TSP, the training dataset configuration follows (Li et al., 2025b). For RL training, self-improvement reward reshaping is adopted. Unless otherwise stated, SL and IL runs use an MTP horizon of 4, and self-speculative decoding is disabled.

D Supplementary Experiments

We first fix the notation of the main result tables and clarify why some of their cells are empty, and then report additional experiments and analyses that extend the main results: an extension to a non-

Table 8: Generalization results for different distributions on TSP-100 (Gap).

	Gaussian	Cluster
PrefixCO-SL (S=32)	0.0004%	0.0118%

Table 9: Comparison on real-world instances from TSPLIB-[50-200] (Gap).

Method	TSPLIB-[50-200]
DIFUSCO (T_s=50) (Sun & Yang, 2023)	0.97%
T2T (T_s=50, T_g=30) (Li et al., 2023)	0.35%
Fast T2T (T_s=10, T_g=10) (Li et al., 2024)	0.28%
GenSCO (C=400, 2Opt) (Li et al., 2025b)	0.061%
PrefixCO-SL (S=32)	0.035%

routing CO problem, generalization to non-uniform distributions and real-world benchmark instances, cross-scale generalization, a discussion on the coupling between the embedder and the decoder, an accounting of parameter counts, controlled comparisons under matched hardware, trajectory, and wall-clock budgets, and further diagnostics for reward shaping and imitation warm-starting.

D.1 Notation and Empty Cells in the Main Result Tables

Tables 1 and 2 compare solvers that follow different decoding protocols, so we summarize the notation here. In both tables, S and G denote the number of sampling and greedy trajectories, aug$\times$8 denotes the eight-fold instance augmentation of POMO, and 2Opt denotes the optional 2-opt post-processing. The remaining protocol-specific knobs follow the original papers: BS k in BQ-NCO is the beam-search width, k in LEHD RRC k is the number of reconstruction iterations, and the diffusion-based solvers keep their own (T_s, T_g) step configurations.

Regarding hardware, all autoregressive baselines in Table 2 were evaluated by us on an A100 GPU, consistent with the inference setup of PrefixCO. The supervised baselines in Table 1 were also evaluated on an A100, with the sole exception of GenSCO, whose reported times come from an RTX 4090 evaluation. We re-evaluate GenSCO on an A100 under the same configuration in Table 14. Since that re-evaluation reaches the same gaps with a longer measured time, the advantage of PrefixCO over GenSCO in Table 1 may be larger than the printed times suggest.

A cell is left empty when no value is available, for one of two reasons. For baselines, the corresponding configuration is either not reported in the original paper, such as the sampling-only configuration of Fast T2T on TSP-100, or absent from the source altogether, such as several GenSCO configurations on TSP-100. In Table 2, an entry is likewise left empty when the solver is not designed for that problem and scale, or when the source reports no result for that configuration. For PrefixCO-SL, the empty cells on TSP-100 at larger budgets are deliberate: the gap already reaches 0.0000% at $G = 8$ and at $S = 8$, so increasing the decoding budget only adds computation time without improving solution quality. We follow the practice of GenSCO and omit such entries instead of reporting them.

D.2 Generalization across Distributions and Real-World Instances

Beyond uniformly distributed instances, we adopt the Cluster and Gaussian distributions from ML4CO-Bench-101 (Ma et al., 2025b) as out-of-distribution evaluation sets for TSP-100. As shown in Table 8, the model trained on uniform instances transfers almost losslessly to both distributions.

We further evaluate on real-world instances from TSPLIB and VRPLIB. Table 9 compares against non-autoregressive solvers on TSPLIB-[50-200] under the decoding protocols used in Table 1, where PrefixCO-SL attains the smallest gap among all compared methods. For VRP, Table 10 reports the result of the RL-trained model on small-scale instances from VRPLIB-[100-200], which reaches a 2.70% gap with a sampling budget of S=512. None of the autoregressive baselines in our comparisons provides results on these instances, so this entry stands alone. It is therefore meant as a check that the model transfers beyond uniformly sampled instances, not as a comparison.

Table 10: Result on small-scale instances from VRPLIB-[100-200] (Gap). None of the autoregressive baselines in our comparisons provides results on these instances.

Method	VRPLIB-[100-200]
PrefixCO (S=512)	2.70%

Table 11: Cross-scale generalization results for TSP (Gap, S=32). Rows denote the training size and columns denote the test size.

Train \ Test	TSP-100	TSP-500	TSP-1000
TSP-100	–	5.8769%	7.089%
TSP-500	0.589%	–	0.089%
TSP-1000	1.119%	0.0006%	–

Table 12: Parameter counts of PrefixCO and the compared baselines.

Method	TSP	CVRP	MIS
PrefixCO	47M	95M	8.4M
POMO (EAS, PO) (Kwon et al., 2020; Pan et al., 2025a)	1.3M	1.3M	–
GenSCO (Li et al., 2025b)	19M	–	11M (RB) / 7M (ER)
DIFUSCO / T2T (Sun & Yang, 2023; Li et al., 2023)	3.9M	–	5.3M
Fast T2T (Li et al., 2024)	3.9M	–	5.3M (RB) / 1.3M (ER)
ReLD (Huang et al., 2025)	–	1.4M	–
LEHD (Luo et al., 2024)	1.4M	1.4M	–
BQ-NCO (Drakulic et al., 2023)	3.1M	3.1M	–

D.3 Cross-Scale Generalization

Table 11 reports the cross-scale generalization results of RL-trained PrefixCO, where each model is trained on one problem size and evaluated on the others without retraining. The generalization pattern is largely inherent to the transformer architecture and follows a trend similar to that of GenSCO (Li et al., 2025b): the model performs well on inference problem sizes ranging from half to twice the training scale, while the performance drops sharply when the scale moves beyond this range.

It is worth emphasizing that a key distinction of our method from previous autoregressive approaches is its capability for end-to-end training on large-scale instances. Even with very limited RL training steps, our approach can quickly adapt to the target problem scale. This makes our framework more generally applicable, because it does not rely on problem-specific generalization properties.

D.4 Discussion on the Coupling between the Embedder and the Decoder

We note that a direct ablation between the LM head and the pointer-style decoder is not possible, because the continuous embedder makes the LM head unusable due to its dependence on a fixed vocabulary, while the pointer decoder is specifically designed to enable continuous embeddings; the embedder and the decoder are inherently coupled. Nevertheless, Table 3 in the main text provides indirect support: the continuous embedder with pointer-style decoding far outperforms the discrete tokenizer with an LM head in both final gap and training loss, and the advantage increases with problem size. Moreover, pointer-style selection, which borrows only the high-level idea of selecting from input elements rather than the original recurrent formulation (Vinyals et al., 2015), is intuitively a more suitable paradigm for CO than an LM head, since it directly selects from the input set.

D.5 Parameter Counts

Table 12 lists the parameter counts of PrefixCO and the compared baselines. For MIS, the notations RB and ER indicate the RB and Erdős-Rényi graph generators, respectively.

Table 13: Effect of reducing the full training schedule to one quarter (Gap, S=32).

	TSP-100	TSP-500	TSP-1000	CVRP-100	CVRP-500
PrefixCO-SL (S=32)	0.0000%	0.0023%	0.0030%	–	–
PrefixCO-SL, 1/4 schedule (S=32)	0.0000%	0.0011%	0.0024%	–	–
PrefixCO (S=32)	0.0002%	0.0093%	–	0.280%	0.956%
PrefixCO, 1/4 schedule (S=32)	0.0003%	0.0211%	–	0.384%	1.110%

Table 14: GenSCO evaluation time re-measured on an A100, with the original 4090 times (Li et al., 2025b) in parentheses.

	TSP-500	TSP-1000
GenSCO (C=40)	30s (18s on 4090)	1.4m (46s on 4090)
GenSCO (C=40, 2Opt)	30s (18s on 4090)	1.4m (58s on 4090)
GenSCO (C=160, 2Opt)	1.9m (1.2m on 4090)	5.4m (3.9m on 4090)

PrefixCO benefits from longer training, but also yields high-quality models even with a limited training budget. To investigate this, we reduce the training schedule of each procedure to one quarter of the original length and evaluate the resulting performance in Table 13. For SL, the shorter schedule slightly improves held-out solution quality, suggesting that the original SL schedule was longer than necessary. For RL, the shortened schedule slightly degrades the gap on the larger instances, indicating that longer RL training still provides a measurable benefit.

D.6 Controlled Comparisons under Matched Budgets

Matched hardware. All autoregressive baselines in Table 2 were evaluated by ourselves on A100 GPUs, consistent with the inference setup of PrefixCO. The SL baselines in Table 1 are also evaluated on A100, with the sole exception of GenSCO, which was originally evaluated on an RTX 4090. To ensure a fair comparison, we re-evaluate GenSCO on A100 under the same configuration, with the detailed times reported in Table 14. To illustrate the implications of these timing differences, consider TSP-500 with C=40: the original paper reports 18 seconds on a 4090, whereas our re-evaluation takes 30 seconds on an A100 with full GPU utilization while achieving the same reported gap. This suggests that the performance advantage of PrefixCO over GenSCO may be even larger than the raw numbers in Table 1 suggest. Since some hyperparameters of GenSCO, such as the batch size, may be hardware-sensitive and require separate tuning on A100, we keep the originally reported results in the main text to avoid underestimating GenSCO.

Matched trajectory counts. Different solvers adopt fundamentally different decoding mechanisms, making a direct one-to-one alignment of trajectory counts hard to realize. Specifically, heavy-decoder autoregressive solvers such as BQ-NCO are configured via beam search width, while LEHD is controlled by the number of RRC iterations, which performs reconstruction rather than full trajectory sampling, so the notion of trajectory count is not directly applicable. For light-decoder autoregressive solvers like POMO, the number of greedy trajectories is inherently determined by the number of starting nodes (i.e., N per instance, or $8N$ with augmentation), and hence is not freely adjustable. Non-autoregressive solvers also have their own distinct configuration protocols. We are nevertheless able to align trajectory counts with light-decoder solvers on TSP-100 and CVRP-100, with the results reported in Table 15.

Matched wall-clock budgets. Aligning wall-clock budgets for all settings is also challenging, since only PrefixCO, GenSCO, and LEHD offer flexible budget control. Accordingly, we align the time budgets for LEHD, GenSCO, and PrefixCO in Tables 16 and 17. Given matched wall-clock budgets, PrefixCO achieves a significantly lower Gap compared with the SOTA baselines LEHD and GenSCO.

Overall, PrefixCO demonstrates substantial improvements over previous baselines, and this conclusion holds even when examining the fastest configurations, PrefixCO (aug$\times$8, G=1) and PrefixCO (aug$\times$8, S=1). Specifically, PrefixCO-SL (aug$\times$8, G=1) delivers roughly a 5-fold reduction in the optimality gap on TSP-1000 while achieving a 16.7$\times$ speedup over the strongest baseline GenSCO (C=160,

Table 15: Comparisons with light-decoder AR solvers under aligned trajectory counts (Gap, Time).

	TSP-100	CVRP-100
POMO aug×8 (Kwon et al., 2020)	0.13%, 8s	1.287%, 8s
POMO+PO aug×8 (Pan et al., 2025a)	0.0425%, 8s	1.181%, 10s
ReLD aug×8 (Huang et al., 2025)	–	1.509%, 7s
PrefixCO (aug×8, G=100)	0.0001%, 4.5m	–
PrefixCO (aug×8, S=100)	0.0001%, 4.5m	0.217%, 5.3m

Table 16: Comparisons with LEHD under matched wall-clock budgets (Gap, Time).

Instance	Method	Gap, Time
TSP-100	LEHD RRC 100	0.0101%, 0.9m
	PrefixCO (aug×8, G=20)	0.0002%, 0.9m
	PrefixCO (aug×8, S=20)	0.0002%, 0.9m
TSP-500	LEHD RRC 100	0.355%, 3.8m
	PrefixCO (aug×8, G=72)	0.0098%, 3.7m
	PrefixCO (aug×8, S=72)	0.0080%, 3.7m
CVRP-100	LEHD RRC 100	0.814%, 1.1m
	PrefixCO (aug×8, S=20)	0.316%, 1.1m
CVRP-500	LEHD RRC 100	2.465%, 3.1m
	PrefixCO (aug×8, S=80)	0.865%, 3.2m

Table 17: Comparison with GenSCO under matched wall-clock budgets on TSP, with GenSCO re-evaluated on A100 (Gap, Time).

	TSP-500	TSP-1000
GenSCO (C=40)	0.016%, 30s	0.258%, 1.4m
GenSCO (C=40, 2Opt)	0.012%, 30s	0.046%, 1.4m
PrefixCO-SL (aug×8, G=8)	0.0033%, 28s	0.0045%, 1.5m
PrefixCO-SL (aug×8, S=8)	0.0053%, 28s	0.0050%, 1.5m

2Opt), with an inference time of just 14 seconds. Regarding comparisons with AR baselines, on TSP-500 PrefixCO (aug×8, G=1) reduces the gap to 14% of that achieved by LEHD RRC 1000 while cutting the solving time from 37.5 minutes to 5 seconds; on CVRP-500, PrefixCO (aug×8, S=1) achieves a 247× speedup over LEHD RRC 1000 while also attaining a lower gap.

D.7 Further Analyses of Reward Shaping and Warm-Start

The reward hacking failure mode is architecture dependent. In matched experiments, we consistently observe the reward hacking failure mode discussed in Sec. 3 for PrefixCO, while it does not occur in POMO under the same training protocol. We therefore interpret it as architecture dependent, rather than a universal failure mode of group-relative RL. This is exactly why non-negative reward shaping is needed: it unlocks the benefits of prefix transformers, namely O(N) decoding and high expressivity, without falling into reward hacking.

Entropy diagnostics. Table 18 tracks the entropy of the policy distribution during RL training with self-improvement reward shaping. The entropy decreases steadily and reasonably over the course of training, indicating that the output distribution is progressively sharpening. However, the reduction is rather modest: the final entropy remains above half of the value observed at the 1/8 training stage (0.0387 vs. 0.0593), which suggests that the model has not collapsed to a degenerate distribution. This controlled sharpening supports our claim that the reward shaping effectively prevents premature convergence while still allowing for improved solution quality. For comparison, without reward

Table 18: Policy entropy during RL training with self-improvement reward shaping, evaluated on TSP-100 (S=32).

Percentage of training completed	1/8	4/8	7/8	8/8
Entropy	0.0593	0.0461	0.0394	0.0387

Table 19: Quality after imitation warm-start on TSP-500. PrefixCO-IL denotes PrefixCO trained via IL on low-quality 2-opt data.

	TSP-500
PrefixCO (S=32)	0.0093%
PrefixCO-IL (S=32)	5.2315%

shaping, the policy entropy becomes unstable almost immediately after RL begins and rapidly diverges, accompanied by the collapse of training. This is consistent with the failure mode observed in Fig. 3: the original group-relative objective induces increasingly pathological probability updates rather than a stable improvement of the overall policy distribution.

Diversity diagnostics. Table 2 shows that with non-negative self-improvement reward shaping, the model can still benefit from the diversity-based multi-trajectory decoding strategy, confirming successful diversity preserving. We attribute this to the inherent complexity of CO combined with an effectively infinite set of training instances, which prevents the model from memorizing a single decision pattern and thereby naturally preserves policy diversity.

Multiple valid continuations at each construction step. In CO, several candidate nodes are usually feasible at the same step, and the decoder of PrefixCO represents this explicitly: the softmax over the feasible candidates in Sec. 3 assigns probability mass to all currently valid next nodes instead of committing to a single mode, and the mask only removes the infeasible ones. The consequence is visible in the decoding results. As the number of sampled trajectories grows from $S = 1$ to $S = 512$ in Table 2, the best-of-S objective keeps improving, which indicates that the policy produces many distinct high-quality solutions rather than repeating one. For CVRP, the starting node is fixed to the depot, so the diversity of these solutions arises entirely from the per-step sampling probabilities. This property concerns the distribution over the next node and is therefore independent of multi-token prediction, which instead predicts several future steps at once.

Necessity of imitation warm-start. IL warm-up is a common practice to address the cold-start problem in RL, that is, the lack of useful signals in the early stages of training due to randomly initialized policies. Its necessity depends on the architecture, task difficulty, reward sparsity, and initialization. While AM uses a 3-layer network and POMO uses a 6-layer network, we scale our architecture to 64 layers for TSP and 32 layers for CVRP, which significantly increases variance and makes IL warm-up indispensable for stable training. Most previous CO methods do not adopt it because they do not encounter early-stage training stagnation. The warm-up data are very cheap to obtain: we use suboptimal solutions generated by the simple 2-opt heuristic, which yield 1.920% and 6.994% gaps for TSP-100 and TSP-500 on the test set. Consequently, the model after IL warm-up only produces more meaningful output probabilities compared to random initialization, but it remains far from the final model quality, as illustrated in Table 19.

D.8 Discussions on MTP and Feasibility Checking

Why MTP improves TSP but degrades CVRP. We mainly attribute this phenomenon to model capacity and constraint complexity. On one hand, CVRP has much more complex constraints than TSP, requiring higher capacity. During training, enabling MTP has a negligible effect on TSP (even a slightly lower average objective in some cases), but for CVRP it causes the objective to rise sharply throughout training, suggesting limited expressivity. On the other hand, our current MTP implementation has a subtle limitation: the decoding queries are linearly correlated, and the dot-product operation expects the decoding query and key to be similar. For TSP, consecutive nodes are close in space, so this works fine. For CVRP, the depot arrival and departure nodes are far apart,

making the linear correlation insufficient and restricting expressivity. For TSP, since the model capacity is sufficient for the constraint complexity, MTP training provides useful long-planning and look-ahead signals, which help the model reason over future steps and improve solution quality. For CVRP, the combination of higher constraint complexity and the linear-correlation limitation outweighs any potential benefit from multi-step prediction, leading to performance degradation. We plan to explore improved MTP designs in future work, and present MTP and SSD as optional exploratory extensions.

Feasibility checking. Regarding feasibility checking, we directly follow previous AR works, since our primary focus is on architectural and training improvements. During training, feasibility is ignored, while during inference it is enforced at generation time through local feasibility checking, which works well for TSP, CVRP, and MIS. For more complex problems such as TSPTW, AR-based methods can potentially fail because invalid partial solutions are common, and since we do not introduce improvements specifically for this aspect, our approach may also face the same limitation. There is a line of specialized research addressing this issue (Bi et al., 2024), but to date no method can guarantee feasibility in all cases, which is why a feasibility rate metric is commonly reported for such problems. These AR plugins are orthogonal to our contributions, and incorporating a more sophisticated verifier would likely require a dedicated study.