SOLVING DIVERSE COMBINATORIAL OPTIMIZATION PROBLEMS WITH A UNIFIED MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Combinatorial Optimization (CO) encompasses a wide range of problems that arise in many real-world scenarios. While significant progress has been made in developing learning-based methods for specialized CO problems, a unified model with a single architecture and parameter set for diverse CO problems remains elusive. Such a model would offer substantial advantages in terms of efficiency and convenience. In this paper, we introduce and formalize a unified model for solving various CO problems. Inspired by the success of next-token prediction, we frame each problem-solving process as a Markov Decision Process (MDP), tokenize the corresponding sequential trajectory data, and train the model using a transformer backbone. To reduce token length in the trajectory data, we propose a CO-prefix design that aggregates static problem features. To address the heterogeneity of state and action tokens within the MDP, we employ a two-stage self-supervised learning approach. In this approach, a dynamic prediction model is first trained and then serves as a pre-trained model for subsequent policy generation. Experiments across nine CO problems demonstrate the generic problem-solving capability of our unified model, highlighting its few-shot and even zero-shot ability to generalize to unseen problems through rapid fine-tuning. We believe our framework offers a valuable complement to existing neural CO methods that focus on optimizing performance for individual problems.

1 INTRODUCTION

Combinatorial optimization (CO) problems are pivotal in a wide range of real-world applications, including logistics and industrial management (Singh & Rizwanullah, 2022). To address these generally NP-hard problems, traditional integer programming and heuristic methods have been extensively studied to obtain either exact or near-optimal solutions over the past decades. With the rapid growth of deep learning, solving CO problems using learning-based methods has garnered increasing attention, giving rise to the field of Neural Combinatorial Optimization (NCO) (Kim et al., 2022; Drakulic et al., 2024). Among all NCO schemes, the auto-regressive construction methods are favored in recent literature (Bello et al., 2016; Kool et al., 2018; Kwon et al., 2020; Kim et al., 2022). These methods construct solutions incrementally, and the entire problem-solving process can naturally be framed as a Markov Decision Process (MDP). These end-to-end methods offer significant computational efficiency and flexibility in generating feasible solutions, as they can easily avoid constraint-violating actions within the MDP framework (Kim et al., 2022).

However, a significant limitation remains: models from existing literature are typically tailored to specific problem types, lacking the ability to handle a wide range of problems simultaneously. There are clear advantages to using a unified model across diverse problems. First, it reduces the need for hand-crafted designs for each individual problem. Second, it facilitates adaptation to unseen problem types more quickly and efficiently than training specific models from scratch. Although some literature claims to propose generic frameworks, these methods generally apply the same general architecture to different problems, but with specific model structures and varying learning parameters. This results in a loss of true generality. The development of these NCO methods aligns with the famous No Free Lunch Theorem (NFLT) (Wolpert & Macready, 1997). Most literature avoids the challenge of achieving generality across different problems, and focuses on improving performances on individual ones, illustrated as Model A and Model B in Figure 1. In contrast, we tackle the challenge of achieving generality across diverse CO problems, posing a new research question: Can we

develop a unified model with a single neural architecture and parameter set that can simultaneously solve diverse CO problems, while maintaining strong few-shot capabilities?

Recently, the concept of next-token-prediction has marked a new era in general artificial intelligence, excelling in processing data across multiple scenarios, domains, and even modalities. The most successful examples are the large language models (LLMs) and multimodal large languange models (MLLMs) (Achiam et al., 2023; Dubey et al., 2024), which can generalize across various natural language process (NLP) and computer vision (CV) scenarios and excel in few-shot learning tasks. Furthermore, the concept has also been applied to decision-making tasks directly (Chen et al., 2021). For instance, Reed et al. (2022) developed a generalist agent to handle different control environments simultaneously, such as Atari games and robot benchmarks. Motivated by these breakthroughs, we explore whether a unified model can be designed to tackle diverse CO problems under the same next-token-prediction framework.

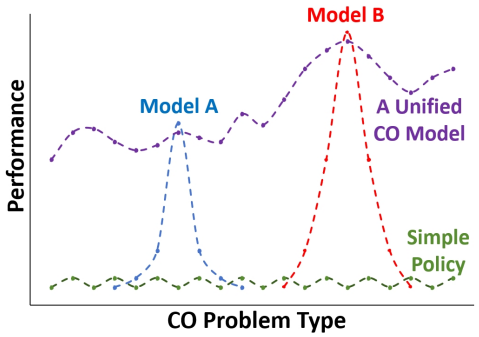


Figure 1: The No Free Lunch Theorem of optimization.

In general, we collect solutions for raw problem instances generated by state-of-the-art solvers from a variety of problem sources. Adopting the widely used auto-regressive MDP formulation from existing literature, we generate optimization trajectories where actions are iteratively selected based on partial solutions. These trajectories are serialized into flat token sequences and trained using a single transformer backbone, as illustrated in Figure 2. However, directly applying existing training schemes to CO problems often proves inefficient. Since most CO problems are NP-hard, the observation space can be large, resulting in long token sequences and reduced training efficiency. Furthermore, a full trajectory contains different types of elements, including states and actions. Predicting all elements in a unified manner, without addressing their distinct roles and the heterogeneity between them, further complicates the training process.

To tackle these challenges, we introduce two approaches to improve generic training performances considering the common characteristics of CO problems. First, we propose a non-causal, decoder-only architecture that incorporates a CO-prefix to reduce the overall token length. Unlike other environments where observations in an MDP can be fully dynamic, most information in a CO problem comes from its static description data. For instance, in a Traveling Salesman Problem (TSP), the distances between node pairs remain unchanged regardless of the visiting order. Therefore, we utilize a CO-prefix to aggregate the problems' static features, while the subsequent main trajectory handles dynamic observations. This reduces token length and improves training efficiency. Second, we decompose the entire token generation process into two self-supervised learning stages to reduce

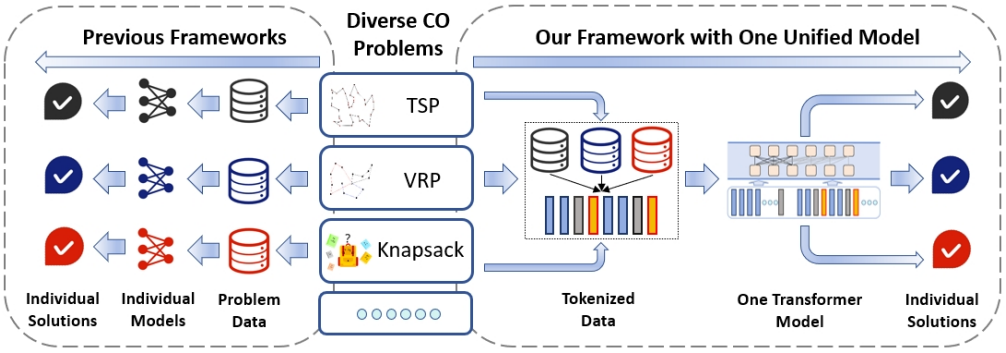


Figure 2: The difference between previous frameworks and ours to solve diverse CO problems. While previous frameworks require individual models with specific designs to adapt to different problems, our framework only utilizes one unified model.

training difficulty. In the first stage, the model focuses solely on learning to predict forward dynamics, which then serves as the pre-trained model for the subsequent policy generation. These two stages are designed to handle the heterogeneous elements within the trajectory, thereby reducing the overall training difficulty.

It is important to note that although one recent literature claims to achieve multi-task learning with cross-problem generalization for vehicle routing problems (VRP) (Liu et al., 2024), it cannot be extended to a unified model as we propose. In their approach, VRPs are formulated as different combinations of shared attributes, such as capacity, backhauls, time windows, duration limits, and open routes. However, the single model designed to solve VRPs within these attribute combinations still relies heavily on human-crafted designs and struggles to generalize to problems outside these specific configurations. In contrast, the framework we propose can be applied to any CO problems, as long as a feasible solution can be formulated as an MDP.

To summarize, our key contributions are:

- To the best of our knowledge, we are the first to thoroughly investigate solving diverse CO problems using a single unified model and to present a corresponding framework. We believe that our framework provides a valuable complement to existing NCO methods that focus on achieving optimal performance for individual CO problems.
- To address the challenges of directly applying existing next-token prediction concepts to CO problems, we introduce a CO-prefix design and a two-stage self-supervised learning scheme to reduce token length and training difficulty.
- We establish a comprehensive testbed featuring nine CO problems to evaluate the generic problem-solving ability of our unified CO model. Experiments show that the model exhibits strong generic problem-solving capabilities. Additionally, we demonstrate its few-shot and even zero-shot generalization abilities when tackling new problems, enabled by fast fine-tuning.

2 Related Works

2.1 Auto-regressive NCO Methods

Auto-regressive NCO methods aim to incrementally build a feasible solution step by step. The pioneering work in this area was the Pointer Network, which was first tested on TSP(Vinyals et al., 2015). Subsequent research extended this idea by incorporating reinforcement learning (RL), demonstrating its effectiveness across a broader range of CO problems (Bello et al., 2016). Routing problems, a significant subclass of CO problems, have been extensively studied within this auto-regressive framework using RL (Kool et al., 2018; Kwon et al., 2020). To better account for both node and edge level features, a matrix-encoding framework was developed (Kwon et al., 2021). The potential of applying auto-regressive NCO methods to more general CO problems was also discussed (Drakulic et al., 2024). These methods offer significant advantages due to their fast inference speed, as their computational complexity during testing remains low. Additionally, they are much more flexible in generating feasible actions that respect various problem constraints.

A recent trend in NCO research is exploring the generalization capabilities of algorithms. Existing methods primarily focus on generalizing across different data distributions (Zhou et al., 2023; Bi et al., 2022) and problem scales (Zong et al., 2022; Li et al., 2021). In terms of generalization to multiple problems, one study attempts to solve various VRPs by decomposing them into several elementary tasks (Liu et al., 2024). However, this decomposition relies heavily on human-designed rules, which limits its generalization potential. To the best of our knowledge, no architecture currently exists for a truly general-purpose unified model capable of addressing a wide range of CO problems.

2.2 Next-Token-Prediction in Decision-Making

In addition to the significant success of next-token prediction in both LLMs and MLLMs, researchers have also explored how to directly incorporate this approach into decision-making problems. Chen et al. (2021) first explored the use of the Transformer (Vaswani, 2017) as an effective backbone for handling various control environments in an offline RL setting, including Atari, OpenAI Gym, and

others. They trained a single policy model to generate actions at each decision step. Janner et al. (2021) further proposed the Trajectory Transformer, which predicts all elements within a trajectory. In addition to offline RL, similar architectures have been integrated with imitation learning (Reed et al., 2022; Shafiullah et al., 2022; Brohan et al., 2022; Zhou et al., 2022). A notable application of this approach is the Generalist Agent, GATO (Reed et al., 2022), which successfully extended its capabilities across multiple control environments using a unified model. Wen et al. (2022) further adapted the GATO structure, referred to as DB1, and extended it to solve TSP problems. Building on these successes, it is natural to consider Transformers as the backbone for a unified model capable of solving diverse CO problems.

However, we note that Wen et al. (2022) employed a pretrained GCN model (Kipf & Welling, 2016) specifically trained for TSP to generate TSP state embeddings. These embeddings were then used to train the unified model, rather than using the original TSP data directly. We believe this approach contradicts the core concept of a unified model, which should rely solely on a single architecture and parameter set. Nevertheless, we adopt the unified model structure proposed by GATO and re-implemented in DB1 as a key baseline for comparison, where only the original trajectory data is processed.

3 METHODOLOGY

3.1 PRELIMINARIES

3.1.1 AUTO-REGRESSIVE MDP FORMULATION FOR CO PROBLEM

We first formulate the sequential construction process of a CO problem solution as an MDP. Following the approach of existing auto-regressive NCO methods (Zhang et al., 2023), a complete solution is incrementally constructed through multiple decision steps.

Let $\mathcal{S}$ denote the entire state space, with states $s_t \in \mathcal{S}$, and let $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ be the action space, where actions are denoted by $a_t \in \mathcal{A}$. All states are assumed to be reachable from the initial state s_1. Since a CO problem is fully observed and deterministic, the transition from state s_t to s_{t+1} is fully determined by action a_t. Each state s_t is represented as a set of actions taken before. A policy in the MDP refers to a distribution $P(s'|s)$ over the states s' that can be reached from from s via a single action. A feasible CO problem solution, represented as a complete trajectory τ, can be further induced by the policy over T steps via $\prod_{t=1}^{T} P(s_{t+1}|s_t)$.

It is important to note that many CO problems exhibit the property of tail recursion: after applying a series of construction steps, the remaining tail subproblem becomes a smaller instance of the original CO problem, as discussed in Drakulic et al. (2024). Any problem with this tail-recursion property can be formulated as the MDP described above. In this paper, we focus on CO problems that exhibit this property.

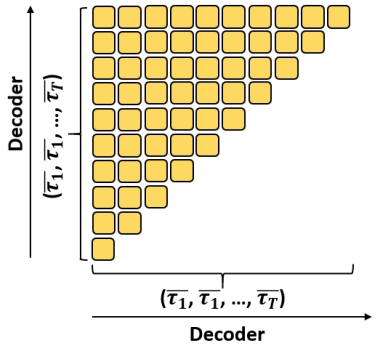
3.1.2 TRAJECTORY DATASETS

To prepare the trajectory datasets for training, we first obtain the final optimized solutions from state-of-the-art solvers for various problems. We then trace their complete optimization MDP episodes, $\tau = (\tau_1, \tau_2, ..., \tau_T)$, where each episode consists of states and actions, with $\tau_t = (s_t, a_t)$ representing the state-action pairs at each step.

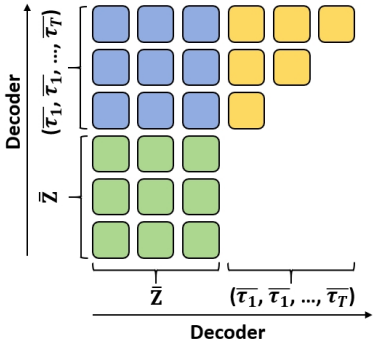
To jointly handle diverse features from different problems and distributions, we flatten all elements within the MDP episode into one dimension and tokenize them through a tokenization process. Discrete values, such as the node indices of actions, are directly assigned with integer token IDs from $[Min_d, Max_d)$. Continuous values, such as demands and positions, are first encoded via mu-law, discretized to N_{bin} uniform bins, and then tokenized into the range $[Min_c, Max_c)$. The final trajectory token sequence $\overline{\tau}$ at each step is formulated with state tokens, followed by an action spliter token $< | >$, and then action tokens:

$$\overline{\tau} = (\overline{\tau_1}, \overline{\tau_2}, ..., \overline{\tau_T}), \quad \text{where } \overline{\tau_t} = (\overline{s_t}, < | >, \overline{a_t}). \tag{1}$$

Note that the length of a fully tokenized sequence can sometimes be excessively long. To address this, we set the target total token length L in advance, and use selected contiguous segments from complete solution MDPs. Additionally, we only preserve dynamic observations in the intermediate progress within s_t, while the static information of the raw problem instances is aggregated within a CO-prefix design, as introduced in the following section. For each problem instance and its complete solution MDP, we collect multiple trajectories as data augmentation. Details of tokenization and trajectory collection can be found in Appendix B.



(a) Causal decoder-only architecture without CO prefix.



(b) Non-causal decoder-only architecture with CO-prefix.

Figure 3: Two architecture designs for the unified model. a) Causal decoder-only architecture without CO prefix, where each token is only conditioned on the past tokens and only trajectory data is processed, adopted in Reed et al. (2022). The entire token length is large. b) Non-causal decoder-only architecture with CO-prefix, where tokens in the CO-prefix shares richer representations conditioned on both prior and past tokens. The trajectory no longer process duplicated static information.

3.2 Non-causal Transformer with CO-prefix

Due to the NP-hard nature of most CO problems, the observation space and dimensionality can be large, resulting in long token sequences and reduced training efficiency.

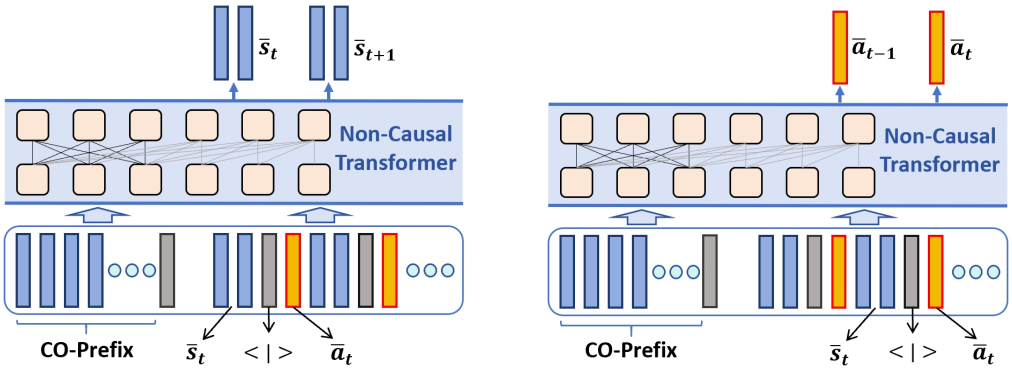
To tackle this challenge, we decompose the original state representation into static and dynamic components, as most of the information in a CO problem comes from its static description data. For instance, in a TSP instance, the positions of the cities are static and remain unchanged throughout the optimization MDP, while the dynamic information only includes the current position. We further introduce a CO-prefix design to capture the static information, which is prepended to the beginning of the token trajectory. The subsequent sequence then focuses solely on dynamic observations. This approach avoids duplicating the representation of observations by tokenizing only the current dynamic state at each step, rather than the entire information. This design significantly reduces token length and improves training efficiency. Let P and $\overline{P}$ represent the raw and tokenized CO-prefix, respectively. The final token sequence fed into the model is $(\overline{P}; \texttt{<X>}, \overline{\tau})$, where $<|>$ denotes a separator token between them.

Although the sequential nature of Markov Decision Processes (MDPs) with time-dependent ordering makes the causal transformer architecture a natural choice due to its simple and effective one-directional design, as suggested in previous sequential decision-making literature (Chen et al., 2021; Reed et al., 2022), shown in Figure 3(b), it has certain limitations. Specifically, the CO-prefix P is time-invariant, as it only contains static representations. Therefore, each token within $\overline{P}$ should be fully visible and processed with each other in a bi-directional manner.

To address this, we adopt a non-causal transformer architecture, where the CO-prefix tokens are processed bi-directionally to ensure comprehensive context integration, while the remainder of the sequence is handled in a one-directional manner, as shown in Figure 3(a).The CO-prefix tokens share richer representations, conditioned on both preceding and subsequent tokens, which enhances overall performance.

Action and CO-prefix Mask To ensure that each action selected by the unified model is feasible during inference, the output policy must be masked to filter out actions that violate problem constraints, using the action mask provided by the problem environment.

It is important to note that during the generation of trajectory data, action masks are collected alongside the trajectory data at each step. During training, the action mask is transformed into the CO-prefix mask, where each token corresponding to an infeasible action is masked in the attention module. For example, in the Traveling Salesman Problem (TSP), the CO-prefix mask includes the coordinates of already visited cities. In the Flexible Flow Shop Problem (FFSP), it corresponds to the job duration entries of completed tasks. This design allows the model to focus on more relevant tokens for feasible actions, without increasing the overall token length.



(a) The dynamics forward stage. (b) The policy generation stage.

Figure 4: Two-stage self-supervised learning to train the unified CO model.

3.3 Two-Stage Self-Supervised Learning

Since a complete trajectory consists of different types of elements, such as observations and actions, predicting them without distinguishing their individual roles further increases the training difficulty.

To address this challenge, we decompose the token generation process into two stages in a self-supervised learning framework: a dynamics forward stage and a policy generation stage, as shown in Figure 4.

- **Dynamics forward stage.** In the first stage, we pre-train the model to predict the next observation given the current action. The training loss for a training batch $\mathcal{B}$ is defined as follows:

$$\mathcal{L}(\theta, \mathcal{B}) = -\sum_{b=1}^{|B|}\sum_{t=1}^{T^b} \log p_\theta(\overline{s_{t+1}^b} | (\overline{Z^b}, I_p, \overline{\tau_1^b}, \overline{\tau_2^b}, ..., \overline{\tau_t^b})), \tag{2}$$

  where T^b is the amount of trajectory units in the current token length. Since MDP transitions are deterministic in CO problems, ths dynamics model can be accurately trained with the same amount of data.

- **Policy generation stage.** In the second stage, we fine-tune the model to generate actions based on the pretrained model in advance. The training loss for a training batch $\mathcal{B}$ is defined as follows:

$$\mathcal{L}(\theta, \mathcal{B}) = -\sum_{b=1}^{|B|}\sum_{t=1}^{T^b} \log p_\theta(\overline{a_{t+1}^b} | (\overline{Z^b}, I_p, \overline{\tau_1^b}, \overline{\tau_2^b}, ..., \overline{\tau_t^b}, \overline{s_{t+1}^b}, I_a)) \tag{3}$$

This two-stage decomposition simplifies the learning process by decomposing the overall process into two sub-tasks, allowing the model to first understand intermediate dynamics and then generate qualified policy. This leads to faster and more effective convergence during training.

4 PERFORMANCE EVALUATION

4.1 PROBLEM AND EXPERT SELECTION

To evaluate the generic problem-solving ability of our proposed framework, we construct a set of nine diverse problems for assessment.

Table 1: The summary of the evaluated CO problems, along with individual expert solver to collect trajectories, the prefix token length and the step state token length. N denotes the number of nodes, items, or jobs, depending on the problem, and M denotes the number of machines in the FFSP.

Problem	Expert Solver	Prefix-Token	State-Token
TSP	LKH3 (Helsgaun, 2017)	$2N$	2
VRP	LKH3 (Helsgaun, 2017)	$3N + 2$	3
OP	Gurobi (Gurobi Optimization, 2018)	$3N + 2$	4
PCTSP	ILS[1]	$4N + 2$	3
SPCTSP	re-opt with ILS	$4N + 2$	3
Knapsack	dynamic programming	$2N$	1
ATSP	LKH3 (Helsgaun, 2017)	$N \times N$	N
MIS	Kamis Lamm et al. (2017)	$N \times N$	N
FFSP	MatNet (Kwon et al., 2021)	$N \times M$	$M + 1$

We first select four common routing problems that have been extensively studied in recent literature (Kool et al., 2018; Kim et al., 2022), including Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), Orienteering Problem (OP) and Prize Collecting TSP (PCTSP). To demonstrate how our model handles uncertainty, we also include the Stochastic PCTSP (SPCTSP). We also consider Asymmetric TSP (ATSP), where the problem is defined on adjacency matrix without Cartesian coordinates (Kwon et al., 2021). Beyond routing problems, we evaluate our model on the Knapsack problem following previous NCO literature Bello et al. (2016); Grinsztajn et al. (2023). We also include the Maximum Independent Set (MIS) problem, which leverages features primarily from graph structures (Sun & Yang, 2023). Finally, we assess our model on the Flexible Flow Shop Problem (FFSP), as suggested by (Kwon et al., 2020).

For each problem, trajectories are collected from individual expert solver, as shown in Table 1.The problem scale is set to $N = 20$, where N represents the number of nodes, items, or jobs, depending on the problem. The instance generation scheme is aligned with previous literature for each problem. Details of data generation and token design can be found in Appendix A.

4.2 EVALUATION PROTOCOLS

Hyperparameters During training, each epoch consists of 400 batches, with 128 trajectories in each batch. The trajectory data for each epoch is newly sampled from a mixed set of all 9 problems. The total token length of each trajectory is $L = 1000$, either clipped or padded from the complete MDP episode data concatenated to the CO-prefix. The transformer architecture uses 10 layers with 768 embedding dimensions. For tokenization, the discrete range is set to $[0, 200)$, the continuous range to $[0, 4]$, and the bin number to 1800. We evaluate the model on the validation dataset every two epochs and apply early stopping if no improvement is observed over 6 consecutive epochs. During inference, performance is evaluated on each problem individually, using a test dataset of 10,000 instances per problem. Further implementation details are provided in Appendix D for reproducibility[2].

Metrics We report four metrics respectively. Following previous NCO literature (Kool et al., 2018), we present the original objectives, the gap from expert results, and the evaluation time on the entire test dataset. Additionally, in line with literature on generic decision-making (Reed et al., 2022), we report performance scores as a percentage, where 100% represents the expert performance for each task, and 0% corresponds to a random policy. The score is calculated as $Score = |obj_e -$

[1] https://github.com/jordanamecler/PCTSP

[2] Our code is available at https://anonymous.4open.science/r/uniCO-35CC/

Table 2: Performance results on all nine problems are presented. The best results among all learning-based models are underlined, and the best results among all unified models are in bold.

	TSP				Knapsack			
Method	Obj.↓	Gap↓	Score↑	Time↓	Obj.↑	Gap↓	Score↑	Time↓
Random	10.47	-	0.00%	(9s)	38.14	-	0.00%	(6s)
Expert	3.84	0.00%	100.00%	(2h)	63.89	0.00%	100.00%	(10m)
POMO-single traj	3.84	0.07%	99.98%	(22s)	63.14	1.17%	97.09%	(30s)
POMO	3.84	0.01%	99.99%	(23s)	63.79	0.16%	99.61%	(31s)
GATO/DB1-greedy	3.99	3.80%	97.68%	(1h)	62.19	2.66%	93.40%	(35m)
GATO/DB1-sampling	3.86	0.49%	99.70%	(15h)	**63.56**	**0.26%**	**98.72%**	(24m)
Ours-DR	3.88	1.04%	99.40%	(8m)	61.78	3.30%	91.81%	(4m)
Ours-greedy	3.87	0.78%	99.55%	(9m)	61.99	2.97%	92.62%	(4m)
Ours-samping	**3.84**	**0.01%**	**99.99%**	(1h)	63.53	0.56%	98.60%	(8h)
	CVRP				OP			
Method	Obj.↓	Gap↓	Score↑	Time↓	Obj.↑	Gap↓	Score↑	Time↓
Random	13.25	-	0.00%	(29s)	1.93	-	0.00%	(8s)
Expert	6.11	0.00%	100.00%	(5h)	5.38	0.00%	100.00%	(1h)
AM-greedy	6.38	4.40%	96.12%	(7s)	5.19	3.72%	93.86%	(9s)
AM-sampling	6.29	2.96%	97.40%	(14m)	5.26	2.55%	95.78%	(7m)
GATO/DB1-greedy	6.63	8.51%	92.72%	(2h)	4.91	8.87%	85.46%	(53m)
GATO/DB1-sampling	6.27	2.41%	97.82%	(18h)	5.30	1.56%	97.42%	(10h)
Ours-DR	6.75	10.47%	91.04%	(15m)	5.00	7.06%	88.99%	(8m)
Ours-greedy	6.66	9.00%	92.30%	(16m)	5.06	5.95%	90.72%	(8m)
Ours-samping	**6.27**	**2.40%**	**97.85%**	(2h)	**5.32**	**1.21%**	**98.01%**	(51m)
	PCTSP				SPCTSP			
Method	Obj.↓	Gap↓	Score↑	Time↓	Obj.↓	Gap↓	Score↑	Time↓
Random	9.25	-	0.00%	(20s)	9.24	-	0.00%	(20s)
Expert	3.16	0.00%	100.00%	(2h)	3.31	0.00%	100.00%	(2h)
AM-greedy	3.18	0.85%	99.57%	(13s)	3.23	-0.71%	101.25%	(9s)
AM-sampling	3.16	0.13%	99.97%	(12m)	3.20	-1.85%	101.94%	(10m)
GATO/DB1-greedy	3.27	3.48%	98.19%	(1h)	3.30	-0.30%	100.17%	(1h)
GATO/DB1-sampling	3.20	1.26%	99.36%	(15h)	3.28	-0.90%	100.47%	(16h)
Ours-DR	3.27	3.48%	98.19%	(13m)	3.28	-0.91%	100.51%	(13m)
Ours-greedy	3.20	1.27%	99.34%	(13m)	3.26	-1.51%	100.84%	(13m)
Ours-samping	**3.15**	**-0.27%**	**100.21%**	(2h)	**3.16**	**-4.03%**	**102.89%**	(2h)
	ATSP				FFSP			
Method	Obj.↓	Gap↓	Score ↑	Time↓	Obj.↓	Gap↓	Score↑	Time↓
Random	10.49	-	0.00%	(10s)	45.00	-	0.00%	(12m)
Expert	3.85	0.00%	100.00%	(2h)	27.31	0.00%	100.00%	(5m)
MatNet	3.87	0.52%	99.70%	(33s)	27.31	0.00%	100.0%	(5m)
MatNet-augment	3.85	0.03%	99.98%	(7m)	-	-	-	-
GATO/DB1-greedy	10.47	171.95%	0.30%	(32m)	41.42	51.67%	20.24%	(4h)
GATO/DB1-sampling	8.86	131.09%	22.78%	(8h)	41.01	50.16%	22.56%	(65h)
Ours-DR	4.38	13.76%	91.87%	(9m)	29.20	6.92%	89.32%	(29m)
Ours-greedy	4.22	9.61%	94.43%	(10m)	29.11	6.59%	89.82%	(27m)
Ours-samping	**3.96**	**3.04%**	**98.15%**	(4h)	**28.34**	**3.77%**	**94.18%**	(7h)
	MIS							
Method	Obj.↑	Gap↓	Score↑	Time↓				
Random	9.11	-	0.00%	(7m)				
Expert	10.44	0.00%	100.00%	(7m)				
LwD	10.42	0.19%	98.50%	(8m)				
GATO/DB1-greedy	9.70	7.09%	44.36%	(33m)				
GATO/DB1-sampling	9.82	5.94%	53.38%	(8h)				
Ours-DR	10.35	0.86%	93.23%	(11m)				
Ours-greedy	10.35	0.86%	93.23%	(10m)				
Ours-samping	**10.42**	**0.19%**	**98.50%**	(1h)				

$obj_r|/|obj - obj_r|$, where obj_e and obj_r denote the objectives of the expert and a random policy respectively (Wen et al., 2022).

Ablation and Baselines We evaluate our proposed model with two variations: with and without the two-stage supervised learning. We refer to the model directly trained to generate actions as *Ours-DR*, as shown in Table 2. For baseline comparisons, we first demonstrate the corresponding

expert approach for each problem as a straightforward benchmark. We then compare our model with GATO (Reed et al., 2022), which was re-implemented and reported by Wen et al. (2022) as DB1. Note that we manually implemented the original GATO framework, as it is not open-sourced. Unlike our approach, GATO is trained using a causal transformer structure, where the trajectory data for each problem is prepended with a prompt sequence from the same problem. The prompt consists of multiple step transitions from other episodes, and other key hyperparameters remain the same as ours. Both GATO and Ours are evaluated using two decoding strategies: greedy decoding and sampling, with 16 solutions per evaluation. Finally, we compare our model with auto-regressive specialist NCO methods, which also use the MDP formulation for CO problems. We report performance on the TSP and Knapsack problems for POMO (Kwon et al., 2020), CVRP, OP, PCTSP, and SPCTSP for AM (Kool et al., 2018), ATSP and FFSP for MatNet (Kwon et al., 2021), and MIS for LwD (Ahn et al., 2020). Note that MatNet is used as both the expert solver and the learning baseline for FFSP. We also report performance for a random policy, along with the evaluation time, which reflects the environment time cost in our implementation.

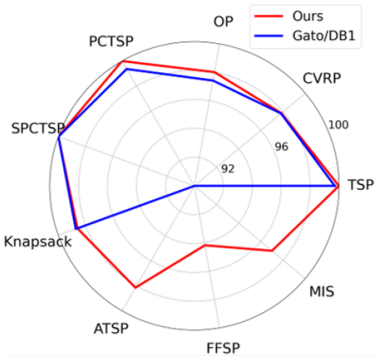


Figure 5: Performances comparison with sampling. Scores larger than 100 are clipped.

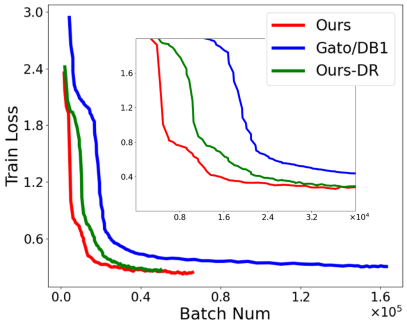


Figure 6: The loss curves along with total batch used of three models during training.

4.3 Performances of Generic Problem Solving

The main evaluation results across all 9 problems are illustrated in Table 2. The best results among all learning-based models, whether specialist or unified, are underlined, while the best results among all unified models are shown in bold. We note that GATO/DB1 struggled to converge effectively on the ATSP, FFSP, and MIS problems under the given evaluation settings. In these cases, the data trajectories of the three problems may have been too noisy for the model to learn other problems effectively. To address this, we trained two versions of GATO/DB1: one on all 9 problems and another on the first 6 problems. We report the better results for the first 6 problems from each model version.

Our unified model demonstrates strong generic problem-solving abilities, achieving performance comparable to specialist models. With greedy decoding, our model achieves scores above 90.7% on all problems except FFSP, and with 16-sample decoding, it reaches 97.8%. Remarkably, when using sampling, our model even outperforms specialist learning baselines under the same setting on 6 out of the 9 problems.

The CO-prefix design is significant. Besides the main table, we also compare the performance of our model with GATO/DB1 in Figure 5. GATO/DB1 struggles to converge effectively on ATSP, FFSP, and MIS, primarily due to its lack of a prefix design. Without this design, GATO/DB1 computes full observation tokens at each step, which becomes highly inefficient when the observation space is large. For instance, in both ATSP and MIS, the static information is represented by the instance adjacency matrix, which has a complexity of $O(N^2)$. In each training episode, GATO/DB1 can only process one or two complete trajectory steps, with or without their prepended prompt sequences. The sparse loss signals from action tokens hinder the model's convergence. Even for the remaining problems, GATO/DB1 converges much slower than our model across all tasks, as shown in Figure 6. Despite this, our unified model still outperforms GATO/DB1 on 5 out of the 9 problems.

The two-stage self-supervised learning scheme improves performances. Compared to a unified model that is directly trained to generate actions, a model fine-tuned on a pre-trained forward dynamics model outperforms across all nine problems when evaluated with greedy decoding. The separation of dynamics prediction and action generation significantly reduces the overall training difficulty, leading to improved solution quality.

4.4 Performances on Few-shot Ability

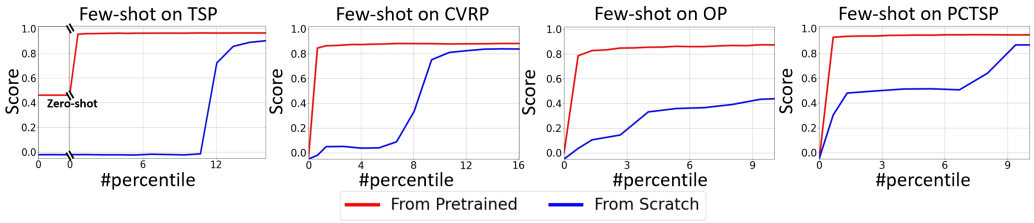


Figure 7: The few-shot results on four routing problems. The x-axis represents the percentage of data used for fine-tuning in relation to the data used in the main results.

To evaluate the few-shot generalization ability of our model on unseen problems, we select four routing problems and train four distinct unified models. Each model is trained in a leave-one-out manner, excluding the selected problem, and then gradually fine-tuned using datasets from the unseen problem. In each epoch for fine-tuning, we use 0.67% of the total data that was used for the problem in the main results. We report the optimization scores and compare them with those of a model trained from scratch on the corresponding problem, as shown in Figure 7.

Overall, our model demonstrates strong few-shot generalization across all four problem settings, even with limited data. In each case, the model achieves high solution quality after just one epoch, using only 0.67% of total data. These results show that our pre-trained unified model can be quickly adapted to an unseen problem with minimal data, eliminating the need for time-consuming retraining of a separate model. This significantly enhances both convenience and efficiency, making it well-suited for real-world applications.

In addition to few-shot abilities, we observed even zero-shot generalization on TSP. The corresponding prefix and step token designs, which only include city coordinates, represent a subset of the more complex routing problems. Our pre-trained model, originally trained on these high-level problems, is able to directly generate solutions with approximately 48% optimality without any additional fine-tuning data.

5 Conclusion and Future Works

In this paper, we have thoroughly investigated the development of a unified model capable of solving a diverse range of CO problems simultaneously. We evaluated the performance of our proposed model on nine different problems, demonstrating that our approach provides a valuable complement to existing NCO methods that focus on optimizing performance for individual CO problems.

As for our future work, we plan to enhance our model to tackle problems with significantly larger token sequences. One promising direction involves integrating our current transformer backbone with Graph Neural Network (GNN)-based structures, as many CO problems are either inherently graph-based or can be reformulated as graph problems. Additionally, we aim to explore incorporating our approach with advances in large model architectures and techniques for efficient long-sequence training..

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International conference on machine learning*, pp. 134–144. PMLR, 2020.

Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems*, 35:31226–31238, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

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, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Paul Erd6s and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:48485–48509, 2023.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018. URL `http://www.gurobi.com`.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

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.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017. doi: 10.1007/s10732-017-9337-x. URL https://doi.org/10.1007/s10732-017-9337-x.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1898–1908, 2024.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

Guman Singh and Mohammad Rizwanullah. Combinatorial optimization of supply chain networks: A retrospective & literature review. *Materials today: proceedings*, 62:1636–1642, 2022.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:3706–3731, 2023.

Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

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

Ying Wen, Ziyu Wan, Ming Zhou, Shufang Hou, Zhe Cao, Chenyang Le, Jingxiao Chen, Zheng Tian, Weinan Zhang, and Jun Wang. On realization of intelligent decision-making in the real world: A foundation decision model perspective. *arXiv preprint arXiv:2212.12669*, 2022.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems*, 36:11952–11969, 2023.

Hang Zhao, Yang Yu, and Kai Xu. Learning efficient online 3d bin packing on packing configuration trees. In *International conference on learning representations*, 2021.

Allan Zhou, Vikash Kumar, Chelsea Finn, and Aravind Rajeswaran. Policy architectures for compositional generalization in control. *arXiv preprint arXiv:2203.05960*, 2022.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pp. 42769–42789. PMLR, 2023.

Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4648–4658, 2022.

A PROBLEM DETAILS

In this section, we continue to introduce the implementation details on each CO problem. We use N to denote either node, item or job amount, and M to denote the total machine amount in FFSP. For each problem, we list the data generation scheme, the expert solver selection, the token (feature) design reference literature, prefix token designs and step token designs respectively. A brief summary is shown in Table 3.

Table 3: The summary of the evaluated CO problems, along with individual expert solver to collect trajectories, the prefix token length and the step state token length. N denotes the number of nodes, items, or jobs, depending on the problem, and M denotes the number of machines in the FFSP.

Problem	Expert Solver	Prefix-Token	State-Token
TSP	LKH3 (Helsgaun, 2017)	$2N$	2
VRP	LKH3 (Helsgaun, 2017)	$3N + 2$	3
OP	Gurobi (Gurobi Optimization, 2018)	$3N + 2$	4
PCTSP	ILS [3]	$4N + 2$	3
SPCTSP	re-opt with ILS	$4N + 2$	3
Knapsack	dynamic programming	$2N$	1
ATSP	LKH3 (Helsgaun, 2017)	$N \times N$	N
MIS	Kamis Lamm et al. (2017)	$N \times N$	N
FFSP	MatNet (Kwon et al., 2021)	$N \times M$	$M + 1$

A.1 TRAVELING SALESMAN PROBLEM (TSP)

In the TSP, the objective is to should find the shortest route that visits each city exactly once and returns to the starting city. The objective is to minimize the total distance of the tour.

Data Generation: We implement the dataset generation scheme described by Kool et al. (2018), for all TSP instances, the positions of N nodes are uniformly randomly sampled in unit square.

Expert Solver: LKH (Helsgaun, 2017).

Token (Feature) Design Reference: AM (Kool et al., 2018), POMO (Kwon et al., 2020).

Prefix Tokens: Coordinates of each city ($2N$ continuous values).

Step State Tokens: Coordinates of the current city (2 continuous values).

Step Action Tokens: The index of the city to visit next.

A.2 VEHICLE ROUTING PROBLEM (VRP)

In the Capacitated VRP (Toth & Vigo, 2014), each city has a certain demand. The objective is to construct multiple routes with minimal a distance that all start and end at a given depot, where the total demands of cities within one route should not exceed the capacity limit. Except for the depot, each city should be visited exactly once.

Data Generation. We implement the dataset described by Nazari et al. (2018). Specifically, each city $i \in \{1, 2, .., N\}$ has a demand $0 < \delta_i \leq D$, where $D > 0$ is the capacity of the vehicle (route). For each route R_j, the total demand of the cities along cannot exceed the vehicle's capacity, i.e. $\sum_{i \in R_j} \delta_i \leq D$. For our experiments, We random sample the location coordinates of the depot and the cities within the unit square uniformly. The discrete demands are sampled uniformly from $\{1, 2, ..., 9\}$ and the capacity is set to $D^{20} = 30, D^{50} = 40$.

Expert Solver: LKH (Helsgaun, 2017).

Token (Feature) Design Reference: AM (Kool et al., 2018), POMO (Kwon et al., 2020).

Prefix Tokens: Coordinates of depot and each city ($2(N+1)$ continuous values), demands of each city (N continuous values).

Step State Tokens: Coordinates of the current location (2 continuous values), current volume budget(1 continuous value).

Step Action Tokens: The index of the location to visit next.

A.3 ORIENTEERING PROBLEM (OP)

In the OP (Golden et al., 1987), each node is assigned with a specific prize. The objective is to construct a single tour that maximize the sum of prizes, starting and ending at a give depot. The tour does not have to include every node anymore, but need to be shorter than a length limit.

Data Generation. We implement the data generation scheme by Fischetti et al. (1998); Kool et al. (2018). Specifically, The location coordinates of depot as well as N node are random sampled uniformly in the unit square. To make the problem more challenging, we made the prize p_i for each node i proportional to its distance from the depot by setting them as:

$$p_i = 1 + \left[99 \cdot \frac{d_{0i}}{\max_{j=1}^{n} d_{0j}} \right], \hat{p}_i = \frac{p_i}{100}$$

where d_{0i} is the distance from node i to the depot. As for the length limit of the route, we set the fixed max length as $T^{20} = 2$ and $T^{50} = 3$, which makes the optimal number of access nodes different from instance to instance.

Expert Solver: Gurobi (Gurobi Optimization, 2018).

Token (Feature) Design Reference: AM (Kool et al., 2018).

Prefix Tokens: Coordinates of the depot and each city ($2(N+1)$ continuous values), prize of each city (N continuous values).

Step State Tokens: Coordinates of the current location (2 continuous values), total prize collected so far (1 continuous value), current length budget (1 continuous value).

Step Action Tokens: The index of the location to visit next.

A.4 PRIZE COLLECTING TSP (PCTSP)

In the PCTSP (Balas, 1989), the sum of total prize is no longer a optimization objective, but a constraint. The objective is to minimize the total route length plus the sum of penalties of unvisited nodes which are given ahead, as well as collecting at least a minimal total prize.

Data Generation. We implement the data generation scheme by Kool et al. (2018). Specifically, as the OP problem mentioned previously, the location coordinates of the depot and all nodes are randomly sampled uniformly within the unit square. For each node i, the associated prize p_i and penalty β_i need to be balanced carefully. If the penalty is too small, the choice of node is almost entirely determined by the total reward constraint; If the penalty is too large, all nodes are always accessed and the total reward constraint fails. Following the reference Kool et al. (2018), we set the prize and penalty as:

$$t_i \sim \text{Uniform}(0, 1), \quad \rho_i = t_i \cdot \frac{4}{N}$$

$$\beta_i \sim \text{Uniform}\left(0, 3 \cdot \frac{K^N}{N}\right)$$

where K^N is about half of the trajectory length of the TSP problem with N cities, we roughly set it as $K^{20} = 2, K^{50} = 3$, and the minimum total prize is set to 1 for our experiments.

Expert Solver: Iterated Local Search (ILS).

Token (Feature) Design Reference: AM (Kool et al., 2018).

Prefix Tokens: Coordinates of the depot and each city ($2(N+1)$ continuous values), prize of each city (N continuous values), penalty of each city (N continuous values).

Step State Tokens: Coordinates of the current location (2 continuous values), prize-to-go to the minimum required total prize (1 continuous value).

Step Action Tokens: The index of the location to visit next.

A.5 STOCHASTIC PCTSP (SPCTSP)

In the SPCTSP, we show how our unified model performs when dealing with uncertainty. Compared to PCTSP, the expected prize of each node is known before the optimization starts, while the real collected prize can only be revealed after visitation.

Data Generation. The data generation for SPCTSP is the sameas in PCTSP, except that we additionally generate the expected prize, which has the same distribution of the real prize.The expert solution algorithm is a modified version of ILS, where the tour is re-optimized iteratively, as suggested by Kool et al. (2018).

Expert Solver: Modified Iterated Local Search (ILS) by suggested Kool et al. (2018).

Token (Feature) Design Reference: AM (Kool et al., 2018).

Prefix Tokens: Coordinates of the depot and each city ($2(N+1)$ continuous values), expected prize of each city (N continuous values), penalty of each city (N continuous values).

Step State Tokens: Coordinates of the current location (2 continuous values), prize-to-go to the minimum required total prize (1 continuous value).

Step Action Tokens: The index of the location to visit next.

A.6 ASYMMETRIC TSP (ATSP)

In the ATSP, the distances between node pairs are no longer determined by Euclidean distances based on node coordinates. Considering a directed graph, the distances are no longer necessarily the same in both directions, and are given in an asymmetric cost matrix upfront. We show how our model performs when dealing with features of $O(N^2)$ complexity.

Data Generation. We follow the same data generation scheme as we did for TSP instances. The cities are selected uniformly in a unit square but only adjacency matrix is visible to represent problem instance.

Expert Solver: LKH3 (Helsgaun, 2017).

Token (Feature) Design Reference: Raw feature usage.

Prefix State Tokens: Adjacency matrix ($N \times N$ continuous values), serialized by rows.

Step Tokens: The row of the current city in adjacency matrix (N continuous values).

Step Action Tokens: The index of the city to visit next.

A.7 KNAPSACK

In the Knapsack problem, a group of items with specific values and volumes are given. The objective is to maximize the total value of items selected without exceeding the total capacity. We designed the problem generation scheme manually and implemented the dynamic programming algorithm for trajectory collection.

Data Generation. We implement a manually designed data generation scheme. Specifically, The values v_i of each item $i \in \{1, 2, ..., N\}$ are randomly sampled as:

$$v_i \sim \text{Uniform}(2, 20)$$

To make the problem more challenging, items of higher value should have a larger volume. We further introduce some randomness and set the volume k_i of item i as:

$$k_i = (1 + t)v_i$$

where $t \sim \text{Uniform}(\{-0.5, 0.5\})$, which means we increase or decrease the volume of item i uniformly and randomly. we set the fixed total capacity as $T^{20} = 30$ and $T^{50} = 75$.

Expert Solver: Gurobi (Gurobi Optimization, 2018).

Token (Feature) Design Reference: POMO (Kwon et al., 2020).

Prefix Tokens: Values of all items (N discrete values), volumes of all items (N discrete values).

Step State Tokens: Current volume budget (1 discrete values).

Step Action Tokens: The index of the newly selected item.

A.8 Maximum Independent Set (MIS)

In the MIS, an independent set is a set of vertices such that no two vertices in the set are adjacent. One should find the largest possible independent set in the graph, meaning it contains the most vertices among all possible independent sets.

Data Generation. We follow the random graph generation scheme proposed by Erd6s & Rényi (1960) , and directly implement the script provided by Sun & Yang (2023) to generate the graphs.

Expert Solver: Kamis (Lamm et al., 2017).

Token (Feature) Design Reference: Raw feature usage.

Prefix Tokens: Adjacency matrix ($N \times N$ discrete values), serialized by rows.

Step State Tokens: Whether each node is selected, excluded or not decided yet. (N discrete values).

Step Action Tokens: The index of the newly selected node.

A.9 Flexible Flow Shop Problem (FFSP)

In the FFSP, N jobs have to be processed in several stages with the same order. Each job in each stage can be handled by a machine from M total machines. The time required for each job at different stages on different machines varies. Each machine can only process at most one job at the same time. The goal is to schedule all jobs so that they can be finished with a minimum of time.

Data Generation. We directly adopt the data generation scheme and script provided by Kwon et al. (2021), where $N = 20, M = 12$. We further implement the corresponding MatNet as the only NCO expert solver in our experiments for trajectory generation.

Expert Solver: MatNet (Kwon et al., 2021).

Token (Feature) Design Reference: MatNet (Kwon et al., 2021).

Prefix Tokens: Job durations in each stage on the corresponding machine of each job ($N \times M$ discrete values).

Step State Tokens: Job durations of the current machine (M discrete values).

Step Action Tokens: The index of the newly selected job to the current machine, or halt.

A.10 Generalization Potential to other CO Problems

n the current stage, we selected nine problems for evaluation. However, our unified model has the potential to be extended to a much broader range of CO problems, particularly those exhibiting the tail-recursion property, as discussed in Section 3. As demonstrated by Drakulic et al. (2024), any problem with this property can be formulated as an MDP. The MDP trajectory data can then be tokenized and processed within our model, thereby equipping the unified model with the ability to solve a wider variety of problems.

We also discuss how our approach could be extended to handle CO problems that are entirely dynamic, such as online bin packing. A key characteristic of such problems is that all relevant features are dynamic. For example, Zhao et al. (2021) proposed maintaining a Packing Configuration Tree (PCT) to hierarchically represent the current packing state. In this MDP formulation, the state at each step includes internal nodes of the PCT, representing the space configurations of packed items, and leaf nodes, representing the potential placements for the current item. The action is the selection of a leaf node.

In this case, the CO-prefix remains empty, as there are no static features to extract, and only step tokens are available. The unified model, therefore, processes MDP transitions without any prior knowledge provided by a CO-prefix. This simplified version of our approach resembles the GATO framework, with only the two-stage training process, and can still perform effectively in problems with relatively small scales, as shown in Table 2 and Table 4, where the token length does not present a significant challenge. However, as token length grows indefinitely, such as in the case of the linearly increasing PCT descriptor in 3D online bin packing, our model may become vulnerable to inefficiencies.

Addressing this challenge aligns with our future research direction, where we aim to improve token usage efficiency and adapt our model to handle larger-scale problems with more tokens.

B Tokenization and Trajectory Collection Details

In this section, we detail the tokenization and trajectory collection methods used in our model.

B.1 Tokenization

A complete trajectory sequence fed into our model consists of two components: the CO-prefix and the subsequent transition steps in the corresponding MDP episode, as illustrated in Figure 8. Both raw CO-prefix P and state s_t at each step contain values that can be categorized into discrete and continuous types, as discussed in the previous section. In most CO problems, the action representation is a discrete value. Both continuous and discrete values are flattened into a one-dimensional sequence and tokenized separately.

- As for continuous values, our goal is to discretize them and map them to unique token IDs. To achieve this, we use mu-law transformation to convert all values into a fixed range. The mu-law transformation is a common technique to handle continuous signals, ensuring that the values are transformed into a finite range suitable for tokenization. The formula for the mu-law transformation is:
$$F(x) = sgn(x)\frac{log(|x|\mu + 1.0)}{log(M\mu + 1.0)} \tag{4}$$
where $M = 4$ and $\mu = 15$ in our experiments, and could be adjusted according to different data distribution. The transformed values are further discretized via $N_{bin} = 1800$ bins, and mapped with token IDs of $\mathbb{Z} \in [200, 2000)$.
- As for discrete values, we directly assign them with token IDs from the integer range $\mathbb{Z} \in [0, 200)$. All discrete values encountered in our previous experiments are strictly less than 200, ensuring that this range is sufficient to cover all discrete values in the data.

In addition to the discrete and continuous values, we also introduce two special tokens for separating key parts of the trajectory sequence.

- Action Splitter: The token <|>, which separates the state tokens from the action tokens at each step, is assigned the token ID 2000.
- Prefix Splitter: The token <X>, which separates the CO-prefix from the subsequent MDP episode, is assigned the token ID 2001.

Once the tokens have been assigned, they are embedded into a continuous vector space using a lookup table. This embedding approach, where each token is mapped to a fixed-length vector, is consistent with the methods used in previous works such as Reed et al. (2022) and Janner et al. (2021). For position encoding, we employ a combination of both local and global position encodings. The local position encoding uses the local index within each step $\overline{\tau}_t$ or the prefix $\overline{P}$, while the global position encoding follows the traditional approach.

B.2 Trajectory Collection and Data Augmentation

In contrast to previous specialist NCO models, which typically use each raw problem instance only once during training or augment it based on symmetries of the CO problem (Kool et al., 2018; Kwon

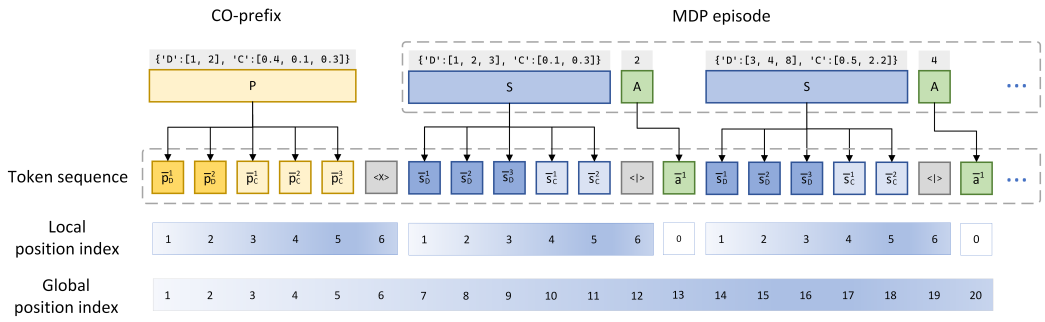


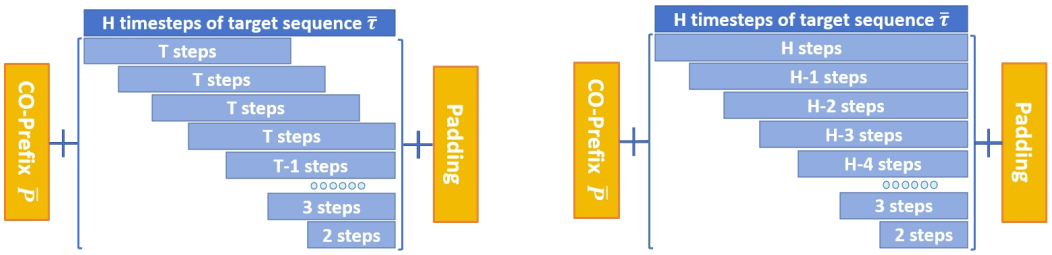
Figure 8: Tokenization illustration of CO-prefix and MDP sequence. 'D' includes all discrete values, and 'C' includes all continuous ones.

et al., 2020), our unified model employs a different data collection strategy. Each raw problem instance, along with its expert solution trajectory, can be used to generate multiple trajectory data for training, either complete or partial, as illustrated in Figure 9.

We set the target total token length L ($L = 1000$ in our main results) in advance, and compute the length of CO-prefix token length for each instance. The remaining token length, which will be allocated to the trajectory data $\overline{\tau}$, is determined by subtracting the CO-prefix token length from the target total token length L. The remaining token length corresponds to the maximum number of time steps H in the target sequence $\overline{\tau}$.

Next, we use the total time steps T from the complete MDP episode and clip subsequences from the original trajectory. If $H > T$, we clip subsequences with steps in the range of $[2, T]$. If $H <= T$, we clip subsequences with steps in the range of $[2, H]$. These subsequences are concatenated to the CO-prefix $\overline{P}$ to form a complete tokenized trajectory. It will be further padded to the target token length L, ensuring that each trajectory can be processed in parallel within a batch. The padded tokens are masked during computation so they do not affect model training.

This approach allows for significant data augmentation, as a single problem instance can generate multiple unique trajectories. Importantly, we do not restrict each trajectory to start from its very first time step during training. Instead, the model learns from the internal transitions between various steps in the trajectory, enhancing its ability to generalize across different stages of the solution process.



(a) $H > T$. (b) $H < T$.

Figure 9: Trajectory collection illustration. Instead of directly using all transitions with T steps of the original MDP episode, we collect subsequences and concatenate them to the prefix $\overline{P}$ as the trajectory data we use for training.

Table 4: Performance results with problem scales of 50. The best results among all learning-based models are underlined, and the best results among all unified models are in bold.

Method	TSP Obj.↓	TSP Gap↓	TSP Score↑	TSP Time↓	Knapsack Obj.↑	Knapsack Gap↓	Knapsack Score↑	Knapsack Time↓
Random	26.08	-	0.00%	(20s)	85.31	-	0.00%	(1m)
Expert	5.69	0.00%	100.00%	(2h)	161.99	0.00%	100.00%	(26m)
POMO-single traj	5.73	0.70%	99.80%	(37s)	161.04	0.59%	98.76%	(1m)
POMO	5.70	0.10%	99.97%	(1m)	161.87	0.08%	99.84%	(2m)
GATO/DB1-greedy	6.25	9.86%	97.22%	(4h)	160.13	0.84%	97.57%	(2h)
GATO/DB1-sampling	5.96	4.64%	98.68%	(62h)	160.63	0.81%	98.20%	(34h)
Ours-DR-greedy	5.99	5.27%	98.53%	(20m)	160.36	1.01%	97.80%	(6m)
Ours-greedy	5.93	4.38%	98.77%	(22m)	160.68	0.81%	98.20%	(7m)
Ours-sampling	**5.78**	**1.45%**	**99.59%**	(3h)	**161.93**	**0.04%**	**99.92%**	(1h)

Method	CVRP Obj.↓	CVRP Gap↓	CVRP Score↑	CVRP Time↓	OP Obj.↑	OP Gap↓	OP Score↑	OP Time↓
Random	30.67	-	0.00%	(1m)	3.14	-	0.00%	(1m)
Expert	10.35	0.00%	100.00%	(12h)	16.59	0.00%	100.00%	(5h)
AM-greedy	10.97	5.88%	97.00%	(20s)	16.01	3.34%	95.84%	(11s)
AM-sampling	10.76	3.79%	98.06%	(35m)	16.55	1.61%	98.01%	(12m)
GATO/DB1-greedy	11.72	12.89%	93.37%	(6h)	14.66	11.57%	85.68%	(3h)
GATO/DB1-sampling	11.19	7.87%	95.96%	(94h)	15.91	4.08%	94.94%	(49h)
Ours-DR-greedy	11.68	12.82%	93.42%	(34m)	15.38	7.29%	91.00%	(17m)
Ours-greedy	11.61	12.14%	93.77%	(34m)	15.49	6.64%	91.77%	(16m)
Ours-sampling	**11.06**	**6.80%**	**96.50%**	(5h)	**16.23**	**2.07%**	**97.44%**	(2h)

Method	PCTSP Obj.↓	PCTSP Gap↓	PCTSP Score↑	PCTSP Time↓
Random	21.37	-	0.00%	(1m)
Expert	4.48	0.00%	100.00%	(5h)
AM-greedy	4.58	2.30%	99.37%	(13s)
AM-sampling	4.53	1.15%	99.69%	(22m)
GATO/DB1-greedy	4.92	9.89%	97.27%	(4h)
GATO/DB1-sampling	4.63	3.23%	99.11%	(65h)
Ours-DR-greedy	4.79	6.92%	98.16%	(29m)
Ours-greedy	4.76	6.30%	98.27%	(29m)
Ours-sampling	**4.54**	**1.36%**	**99.63%**	(4h)

C Additional Results

C.1 Supplementary Performances on Larger Scales

In addition to the main results where $N = 20$ for all problems, we further evaluate the performance of our unified model on larger problem scales. Specifically, we examine a problem scale of $N = 50$ for five selected problems, and summarize the results in Table 4.

The results demonstrate that our proposed unified model maintains consistent performance even as the problem scale increases from $N = 20$ to $N = 50$. Notably, our model outperforms the GATO/DB1 baseline and achieves performance comparable to that of single-model baselines. Our model even outperforms the POMO baseline on the Knapsack problem. These results underscore the robustness and scalability of our unified model, confirming that it is capable of handling problem instances with larger scales while maintaining high-quality performance across diverse CO problems.

C.2 Generalization to Larger Scales

In addition to evaluating our unified model on test sets of the same scale as the training set, we further analyze how well the model generalizes to larger-scale problems. To do so, we utilize the pre-trained model that was trained and reported in Table 2 from Section 4. We then fine-tune this model on newly collected trajectory data for TSP with larger problem sizes: $N = 100$ and $N = 200$. The fine-tuning is performed for 10 epochs for each scale, and the results are compared with the POMO baseline (Kwon et al., 2020).

The performance results are shown in Figure 10, where we observe how the model adapts to larger problem sizes. Results demonstrate that POMO, as a specialist model, can be directly generalized to large scale problem even without finetuning. The unified model still requires finetuning steps to re-obtain problem solving ability. However, the necessary finetuning is fast. After only 3 and 5 epochs each, our unified model outperforms POMO already. These results highlight the model's ability to scale effectively and provide valuable insights into the impact of fine-tuning on performance as the problem size increases.

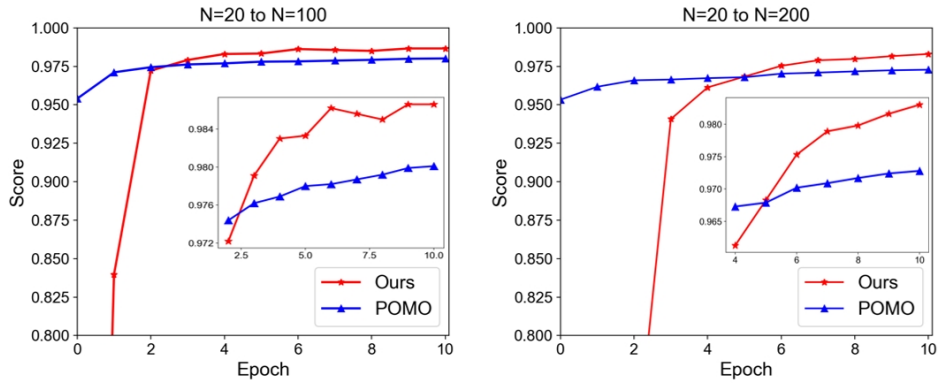


Figure 10: Results of finetuning the unified model trained with $N = 20$ problems in Table 2 to large scale TSP problem with $N = 100$ and $N = 200$.

Table 5: Performances comparison on different problem combinations. Three combinations are considered: all 9 problems, 6 routing problems and 3 non-routing problems. The best results are highlighted in bold.

	All Problems		Routing Problems		Non-Routing Problems	
	Obj.	Score	Obj.	Score	Obj.	Score
TSP	**3.87**	**99.55%**	4.03	96.75%	-	-
CVRP	**6.66**	**92.30%**	6.89	88.72%	-	-
PCTSP	**3.20**	**99.34%**	3.38	96.25%	-	-
OP	**5.06**	**90.72%**	4.74	80.02%	-	-
SPCTSP	**3.26**	**100.84%**	3.39	98.55%	-	-
ATSP	4.22	94.43%	**4.11**	**95.80%**	-	-
Knapsack	**61.99**	**92.62%**	-	-	61.95	92.47%
MIS	**10.35**	**93.23%**	-	-	10.28	87.97%
FFSP	**29.10**	**89.88%**	-	-	29.11	89.85%

C.3 Analysis on Problem Combinations

To better understand how the combination of different CO problems influences the performance of our unified model, we train the model on three distinct problem groups: (1) all nine problems, (2) six routing problems, and (3) three non-routing problems. The performance results are evaluated via greedy decoding, and are shown in Table 5.

Interestingly, we find that aggregating problem instances from structurally diverse problems can further boost the overall performance of the model. Except for ATSP, training on all nine problems together results in the best scores across all other problems compared to the other problem group combinations.

This observation demonstrates the effectiveness of a unified model trained on a diverse set of problems, as it can continuously improve its performance even as the data and problem types become more varied. This phenomenon aligns with findings from GATO (Reed et al., 2022), where the model showed advantages when trained across different tasks, and we further confirm its applicabil-

ity to combinatorial optimization problems. Our results provide compelling evidence that a unified model can generalize well across a wide range of CO problems.

C.4 Ablation on Parameter Scales

To better understand the effect of parameter scale on overall performance, we train several versions of our model with different parameter scales on five problems with $N = 50$. Specifically, we focus on adjusting the width of the transformer backbone, i.e., the embedding dimensions. The results of these experiments are summarized in Table 6.

We observe that the performance of our model continues to improve as the total parameter scale increases. However, the rate of improvement gradually slows down when the total parameter scale reaches 75M and 131M, corresponding to embedding dimensions of 768 and 1024, respectively. Among these configurations, the model with 131M parameters outperforms the model with 75M parameters on three out of five problems.

While increasing the parameter scale generally improves performance, we find that further scaling the parameters beyond a certain point yields diminishing returns. This suggests that the current limitations are not solely related to parameter scale but may also be influenced by the number of problem types and the amount of data used for training. Moving forward, we aim to further explore how increasing the diversity of problem types and expanding the data size can enhance the scalability of our model, unlocking its full potential.

Table 6: Ablation study on different embedding dimensions. The best results are in bold.

	h=128 #params=2.7M		h=256 #params=9M		h=512 #params=34M		h=768 #params=75M		h=1024 #params=131M	
	Obj.	Score	Obj.	Score	Obj.	Score	Obj.	Score	Obj.	Score
TSP	6.82	94.44%	6.02	98.37%	5.94	98.78%	5.96	98.66%	**5.92**	**98.83%**
CVRP	12.55	89.13%	11.75	93.12%	11.59	93.87%	**11.59**	**93.89%**	11.68	93.41%
OP	11.22	60.07%	15.18	89.41%	15.55	92.16%	15.37	90.76%	**15.61**	**92.62%**
PCTSP	5.56	93.30%	4.91	97.33%	4.77	98.20%	4.81	98.00%	**4.71**	**98.59%**
Knapsack	140.14	69.94%	160.28	97.69%	160.28	97.65%	**160.49**	**97.96%**	160.36	97.80%

D EVALUATION DETAILS AND TRAINING PROCESS REPORTS.

In this section ,we provide more implementation details for reproducibility.

Table 7: Implementation details.

Module	Element	Detail
System	OS	Ubuntu 22.04.2
	CUDA	11.7
	Python	3.11.4
	Pytorch	2.0.1
	Device	2*NVIDIA A100 80G
Hyperparameters	Backbone	Llama
	Embedding dimension	768
	Layer Num	10
	Q Head Num	8
	KV Head Num	8
	Total token length L	1000
	RMS Norm epsilon	1e-6
	Weight Decay	1e-4
	Early Stopping Runs	6
	M of μ-law	4
	μ of μ-law	15
	$[Min_d, Max_d)$	$[0, 200)$
	$[Min_d, Max_d)$	$[200, 2000)$
	Optimizer	AdamW
	inital learning rate	0
	max learning rate	2.5e-4
	leanring rate warmup ratio	5%
	leanring rate decay ratio	75%
	leanring rate decay factor	10
	leanring rate decay style	cosine