Keep Searching, the Optimum Is Out There!

Anonymous Author(s)
Affiliation
Address
email

Abstract

Achieving robust *anytime performance* in combinatorial optimization requires a delicate balance between exploration and exploitation. Effective strategies in the optimization literature typically involve: (i) evolving multiple search trajectories for each instance, (ii) preserving diversity among them, and (iii) exploring the entire search space in a structured manner. Most Neural Combinatorial Optimization (NCO) methods, however, rely on either simple constructive heuristics or single-trajectory local search, causing them to underperform when compared to state-of-the-art methods. In this work, we introduce a collaborative multi-agent system that effectively balances exploration and exploitation, bridging the gap between learning-based and classical methods, and achieving unprecedented anytime performance in NCO. Each agent iteratively refines a candidate solution while sharing information through a centralized memory, promoting both cooperation and search diversity. When an agent becomes trapped in a local optimum, it discards its current solution and uses a conditioned constructive network to generate a new, high-quality solution that differs from those of other agents. Empirical evaluations on multiple binary combinatorial benchmarks, including Maximum Cut and Maximum Independent Set, show that our framework achieves superior anytime performance compared to existing NCO methods and improves upon the state-of-the-art specialized solvers.

1 Introduction

Neural Combinatorial Optimization (NCO) [Bengio et al., 2021, Mazyavkina et al., 2021, Bello et al., 2016] represents an emerging framework that aims to address Combinatorial Optimization Problems (COPs) through Neural Networks (NN) in an end-to-end manner. The core premise of NCO is its ability to perform a *train-once, infer-multiple-times* approach, where a trained NN model can generalize to unseen problem instances without necessitating retraining for each new instance. Making NCO approaches suitable for scenarios where rapid inference over many instances is required, such as online decision-making systems [Luo et al., 2024], real-time applications [Cappart et al., 2023], or large-scale optimization pipelines [Zhou et al., 2024].

Early NCO approaches focused primarily on constructing solutions from scratch using neural networks to build approximate solutions in an autoregressive fashion. These methods, known as *Neural Constructive* (NC) methods [Vinyals et al., 2015, Bello et al., 2016, Kool et al., 2018, Kwon et al., 2020], initiate the process with an empty structure and incrementally add one component at a time until a complete and feasible solution is formed. Although effective, NC methods are inherently limited by the construction of a single solution: they commit early to substructures without the ability to revise decisions [Sun et al., 2024]. This myopic search behavior limits exploration, reduces solution diversity, and hampers scalability to larger or more complex instances [Son et al., 2025].

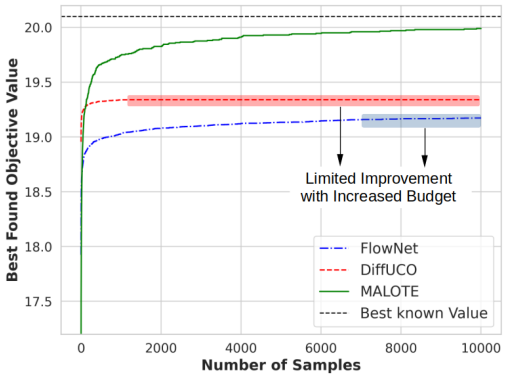
To mitigate the exploration limitations of NC methods, the research community has developed *Neural Improvement* (NI) methods [Chen and Tian, 2019, Lu et al., 2019, Barrett et al., 2020, Wu et al., 2021].

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

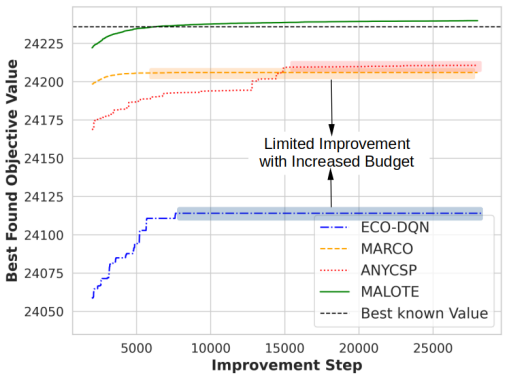
Inspired by classical local search heuristics [Blum and Roli, 2003], NI methods use NNs to iteratively refine a single candidate solution by modifying specific components. Although this enables exploring the local neighborhood, the search often remains confined to a narrow region: once a promising area is identified, the method tends to exploit it excessively, limiting broader exploration [Garmendia et al., 2023].

These issues in NCO and the success of classical optimization techniques highlights a critical gap. The most successful combinatorial optimization algorithms, such as metaheuristics [Blum and Roli, 2003] and exact solvers [Gurobi Optimization, LLC, 2023], often rely on three key principles: (i) maintaining a diverse population of candidate solutions [Mitchell, 1998, Bonyadi and Michalewicz, 2017], (ii) promoting diversity among individual search agents [Baste et al., 2022, Ren et al., 2024], and (iii) systematically exploring the search space in a structured way [Modaresi et al., 2020, Jooken et al., 2023]. Together, these strategies enhance the robustness of the search process and help prevent premature convergence to suboptimal regions.

Yet, most existing NCO methods do not incorporate these principles, leading to limited *anytime performance*, i.e., the ability to improve solutions over time. As shown in Figure 1, previous NCO approaches struggle to further improve these solutions when given additional computational budgets.



(a) NC methods (FlowNet, DiffUCO) on 500 RB graphs (200–300 nodes) for Maximum Independent Set.



(b) Baseline NI methods (ECO-DQN, MARCO, ANYCSP) on 128 ER graphs (700–800 nodes) for Maximum Cut.

Figure 1: Average objective values of the best solutions found by (a) NC and (b) NI methods. Plateaus indicate periods of stagnation despite continued computation. Results include our method (detailed later) to illustrate the performance gap motivating this work.

Our goal is to advance the anytime performance of NCO models by introducing more sophisticated techniques capable of outperforming current state-of-the-art designs. To this end, we propose *Multi-Agent Learning for Optimization with Trajectory Exploration* (MALOTE), a collaborative multi-agent reinforcement learning framework. MALOTE deploys multiple search agents in parallel, each improving a candidate solution while sharing information through a centralized memory that prevents revisiting previously explored solutions. To enhance exploration of the search space, each agent is equipped with an exploration module that activates when progress stalls in a non-improving region. Upon activation, the exploration module discards the agent's current solution and generates a new, high-quality candidate that is explicitly diverse from those of other agents, using a conditioned neural constructive network.

MALOTE is designed to be broadly applicable across different combinatorial optimization problems. In this work, we validate its effectiveness on two representative problems, Maximum Cut (MC) [Dunning et al., 2018] and Maximum Independent Set (MIS) [Lawler et al., 1980]. We also provide practical guidelines for extending MALOTE to alternative problems. Experimental results show that MALOTE not only surpasses previous state-of-the-art problem-specific algorithms on several benchmarks, but also exhibits strong generalization, achieving these results despite being trained on smaller graphs.

The main contributions of this paper are as follows. (a) We introduce *Multi-Agent Learning for Optimization with Trajectory Exploration* (MALOTE), a novel multi-agent framework augmented

with a shared memory component. (b) We design a dual-module agent architecture comprising a first module for iterative improvement of candidate solutions, and a second module to enable global exploration by restarting the search when agents get stuck in non-improving regions. (c) We introduce a novel conditioned Neural Constructive (cNC) network, that generates new solutions conditioned by an exploration weight, which dynamically adjusts the emphasis between optimizing solution quality and exploring diverse solution spaces. (d) We demonstrate that a single trained model can generalize across graph sizes and distributions, highlighting strong transfer capabilities. (e) We validate MALOTE on two classic combinatorial optimization problems: Maximum Cut and Maximum Independent Set. (f) We conduct comprehensive evaluations by benchmarking MALOTE against exact methods, heuristics, metaheuristics, learning-based approaches, and ablation studies to assess the impact of each component.

2 Related Work

In this section, we review the main contributions in NCO, structured around two core methodological families: Neural Constructive (NC) and Neural Improvement (NI) approaches. We mention the key limitations of these methods and examine how recent developments have addressed them with memory-based strategies and multi-agent architectures.

Neural Constructive methods. Initial NC proposals by Vinyals et al. [2015] and Bello et al. [2016] introduced the Pointer Network, which constructs approximate solutions for the Traveling Salesperson Problem using Supervised Learning (SL) and Reinforcement Learning (RL), respectively. Subsequently, Khalil et al. [2017] proposed the structure2vec (S2V) architecture, capable of representing the graph structure of various COPs and training it with Q-learning. These were followed by a large number of incremental works, which extended the NC pipeline to different problems, such as the Maximum Cut [Barrett et al., 2022], the Maximum Independent Set [Ahn et al., 2020] and Job Shop Scheduling Problem [Zhang et al., 2020]; or using improved neural architectures, such as Graph Neural Networks [Cappart et al., 2023] or Transformers [Bresson and Laurent, 2021]. Moreover, apart from SL and RL paradigms, the NCO field has recently seen a growing interest in generative sampling frameworks based on Unsupervised Learning (UL), particularly through diffusion probabilistic models [Sun et al., 2023, Sun and Yang, 2023, Sanokowski et al., 2024].

Despite these advances, previously proposed NC methods remain limited due to their inherently Markovian structure, which ignores information from previously generated solutions. As a result, they are prone to redundancy, often producing similar or even duplicate outputs.

Neural Improvement Methods. Common architectures used in NI include Long Short-Term Memory (LSTM) networks [Chen and Tian, 2019], GNNs [Barrett et al., 2020] and attention-based models [Lu et al., 2019, Hottung and Tierney, 2020].

For instance, Chen and Tian [2019] use an LSTM to score regions for modification and select the corresponding rules. Lu et al. [2019] apply attention mechanisms to choose local operators for the capacitated vehicle routing problem. Similarly, Hottung and Tierney [2020] propose a large neural neighborhood search that uses attention to destroy and repair solution segments. Wu et al. [2021] train policies to select node pairs for applying local operators like 2-opt for routing problems, while da Costa et al. [2020] extend this approach to k-opt operators.

As the mentioned NC works, these NI approaches lack mechanisms to account for previously visited solutions, making them susceptible to repetitive cycles and revisiting identical or similar solutions [Garmendia et al., 2024]. This shortcoming makes it difficult for NI approaches to outperform classical metaheuristics [Blum and Roli, 2003], which typically use mechanisms such as tabu lists or population diversity to prevent revisiting the same solutions. To address this issue, recent work in NCO has introduced explicit memory components to track and avoid previously explored solutions.

Memory-based Neural Methods. ECO-DQN [Barrett et al., 2022], an improvement-based framework, tracks recent action histories to prevent immediate backtracking on solution modifications. Similarly, MEMENTO [Chalumeau et al., 2024] extends NC methods by maintaining a repository of past solutions and consulting this memory to avoid redundant exploration. Although these schemes effectively reduce cyclic behavior along a single search trajectory, their reliance on one trajectory often restricts exploration to narrow regions of the solution space. More expansive and diverse

exploration can be achieved by deploying multiple concurrent search trajectories, as realized in multi-agent paradigms. Building upon these, MARCO Garmendia et al. [2024] introduces a memory module that stores entire solutions along with their corresponding actions, while punishing the model whenever it proposes a repetitive move during training.

Multi-Agent Systems. Some multi-agent methods fall under this broader taxonomy, although they typically consist of agent populations where each agent is specialized for different types of problem instances, rather than collaboratively tackling the same instance. For example, Poppy [Grinsztajn et al., 2023] trains a population of RL agents, each tailored to a specific class of instances. PolyNet [Hottung et al., 2024] generalizes this concept by using a single model conditioned on an input vector to select among multiple policies.

Closer to collaborative approaches are methods that integrate neural networks into classical population-based metaheuristics. DeepACO [Ye et al., 2024], for instance, combines neural networks with Ant Colony Optimization (ACO) [Dorigo and Stützle, 2019], using learned heuristics to guide the search and refine solutions. While ACO is inherently population-based, DeepACO primarily functions as a neural enhancement layered on top of a conventional algorithm, rather than a fully end-to-end population-based neural optimization method.

MARCO [Garmendia et al., 2024] can also be interpreted as a multi-agent system, where a parallel population of NI agents tackles the same instance. However, these agents are not trained collaboratively: each maintains its own memory without access to shared information. As a result, exploration remains localized, lacking global coordination or broader search strategies.

3 Problem Formulation

We consider a combinatorial optimization problem defined on an instance $\mathcal{I}$, consisting of a set of elements V to be optimized under certain constraints and objectives. Let $\mathcal{S}$ denote the set of all feasible solutions for instance $\mathcal{I}$. Each solution $\mathbf{s} \in \mathcal{S}$ has an associated objective value $f(\mathbf{s})$, which measures its quality. The goal is to find the *optimal solution*:

$$\mathbf{s}^* = \arg\max_{\mathbf{s}\in\mathcal{S}} f(\mathbf{s}),$$

where the maximization (or minimization, depending on the problem) is performed over all feasible solutions in $\mathcal{S}$.

4 MALOTE

In this section, we introduce **MALOTE** (*Multi-Agent Learning for Optimization with Trajectory Exploration*), a multi-agent reinforcement learning framework [Busoniu et al., 2008, Albrecht et al., 2024] designed for NCO. A general overview of the framework is shown in Figure 2.

Formally, we define a population of RL agents $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$, where each agent A_i maintains a candidate solution $\mathbf{s}_i \in \mathcal{S}$.

In each iteration t, an agent A_i proposes an action that modifies its current solution $\mathbf{s}_i^t$ to produce a (potentially better) new solution $\mathbf{s}_i^{t+1}$. Depending on the specific combinatorial problem, these actions may include operations such as adding, removing, swapping, or flipping the elements or their values within the solution, or even replacing entirely the candidate solution with a new one. These new solutions are stored in a shared memory $\mathcal{M}$ accessible by the entire population (see the left section of Figure 2).

To determine the action, each agent receives a representation of the current optimization process, referred to as the state $\mathbf{x}^t$. The state $\mathbf{x}_i^t$ of agent A_i comprises:

- $\mathcal{I}$: **Problem Instance.** Includes the complete problem description, such as the graph structure (adjacency matrix) and node- or edge-features, which define all constraints and parameters.
- $\mathbf{s}_i^t$: **Current Candidate Solution.** Represents the current attempt of agent A_i in the solution space.

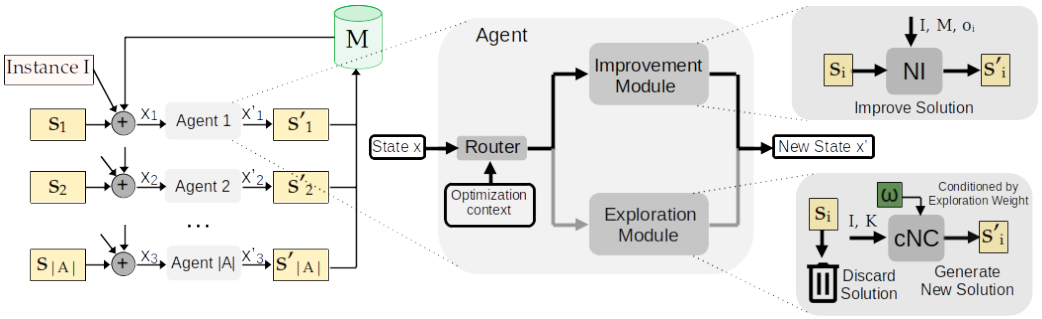


Figure 2: Pipeline of MALOTE. Each agent i receives a state $\mathbf{x}_i^t$, comprising the problem instance $\mathcal{I}$, current solution $\mathbf{s}_i^t$, memory-based historical data $\mathcal{M}$ and the optimization context $\mathbf{o}_i^t$. A router selects the appropriate module: the *improvement* module refines $\mathbf{s}_i^t$ using a neural improvement model, while the *exploration* module replaces it with a new solution generated by a conditioned neural constructive model. This model balances quality and diversity with respect to a memory subset $\mathcal{K}$, guided by an exploration weight ω. The resulting solution $\mathbf{s}_i^{t+1}$ is stored in memory and integrated into the updated state $\mathbf{x}_i^{t+1}$.

- $\mathcal{M}$: **Shared Memory.** A centralized repository for storing all solutions explored by the agents, enabling shared learning and avoidance of duplicate work.
- $\mathbf{o}_i^t$: **Optimization Context.** Contains agent A_i's additional, time-specific details (e.g., past performance or number of non-improving moves) that help adjust its strategy.

To effectively balance exploration and exploitation during the search, each agent is equipped with two specialized modules: an **improvement module** and an **exploration module**. A central component called the **router** dynamically decides which module to activate, depending on the current optimization context (see the center section of Figure 2). The improvement module applies a NI policy to refine the current solution, aiming to incrementally improve its quality. However, if the progress made by the NI policy stalls, the exploration module replaces the current solution with a new one via a NC policy. This new solution is generated based on a trade-off between quality and diversity, controlled by a conditioning weight (see the right side of Figure 2).

We elaborate on the functionalities and mechanisms of the modules and the router in the following sections. While we present our approach in the context of binary optimization problems, it is generalizable to other problem types. In Appendix B, we discuss how it can be adapted to permutation-based problems.

4.1 Improvement Module: Memory-based Neural Improvement Method

The memory-based NI policy, π_{NI}, refines solutions by taking an input $\mathbf{x}_i^{\text{impr}} = (\mathcal{I}, \mathbf{s}_i, \mathcal{M})$, consisting of the problem instance, agent i's current solution $\mathbf{s}_i$, and shared memory information $\mathcal{M}$. It produces an action $a_i \in \mathcal{A}$ to improve the solution as follows:

$$\mathbf{s}_i' = \text{ApplyAction}(\mathbf{s}_i, a_i).$$

For binary problems, the ApplyAction function flips a specific bit. Formally, given a solution $\mathbf{s}_i = (s_{i1}, s_{i2}, \ldots, s_{in}) \in \{0,1\}^n$, action a_i corresponds to selecting an index $j \in \{1, \ldots, n\}$, indicating the bit to flip. The updated solution $\mathbf{s}_i'$ is thus:

$$s_{ik}' = \begin{cases} 1 - s_{ik}, & \text{if } k = j, \\ s_{ik}, & \text{otherwise.} \end{cases}$$

The **training** of the improvement policy employs a reward function with two components: an improvement reward and a repetition penalty.

The *improvement reward* captures the increase in the objective value of a candidate solution. Instead of assigning a reward for every action based on its immediate effect on the objective value, following

recent NI approaches [Barrett et al., 2020, Chalumeau et al., 2024], we use the best solution achieved by each agent, denoted as $\hat{\mathbf{s}}_i$, as a reference point. The improvement reward is only given when an agent discovers a solution that surpasses this baseline, and zero otherwise.

$$R^i_{\text{obj}} = \max[f(\mathbf{s}'_i), f(\hat{\mathbf{s}}_i)] - f(\hat{\mathbf{s}}_i). \tag{1}$$

This ensures that the cumulative reward that an agent receives throughout its entire trajectory corresponds to the total improvement in the objective value.

The *repetition penalty* discourages agents from visiting solutions already present in the shared memory $\mathcal{M}$. For agent A_i, the penalty is defined as:

$$R^i_{\text{rep}} = \begin{cases} -1 & \text{if } \mathbf{s}'_i \in \mathcal{M}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The total reward is calculated as $R_{\text{NI}} = R^i_{\text{obj}} + w_{\text{rep}} \times R^i_{\text{rep}}$ where w_{rep} is the repetition penalty weight. For a complete description of the training methodology for the NI model, refer to Appendix C.

4.2 Exploration Module: Conditioned Networks for Balancing Quality and Diversity

The Exploration Module aims to prevent premature convergence by maintaining a diverse set of solutions in the population. To achieve this, we employ Conditioned Networks [Abels et al., 2019], a neural architecture originally developed for multi-objective reinforcement learning tasks [Felten et al., 2024]. Conditioned networks incorporate conditioning variables as context into their input, enabling the network to dynamically adjust its policy based on external parameters or specific objectives.

Specifically, we propose a conditioned Neural Constructive (cNC) policy π_{cNC} that, given an instance $\mathcal{I}$, a fixed subset of previously visited solutions $\mathcal{K} \subseteq \mathcal{M}$, and an exploration weight $\omega \in [0, 1]$ as part of its input, generates a new candidate solution balancing two competing goals: solution quality and diversity with respect to previously visited solutions. Here, ω directly modulates this balance, with higher values promoting exploration (generating more diverse solutions) and lower values favoring exploitation (producing higher-quality solutions).

We use a fixed-size subset $\mathcal{K}$ instead of the dynamic memory to maintain consistent input dimensions and computational tractability. $\mathcal{K}$ contains exactly the most recent solution from each agent, ensuring $|\mathcal{K}| = |\mathcal{A}|$.

Given input $\mathbf{x}^{\text{explor}}_i = (\mathcal{I}, \mathcal{K}, \omega)$, the policy π_{cNC} outputs a heatmap indicating preference scores for each item's inclusion. This heatmap is decoded into a solution using a Non-Autoregressive (NAR) approach [Joshi et al., 2020], which constructs the solution in a single step by greedily assigning the highest-valued choice for each item. Compared to autoregressive decoding, requiring $O(|V|)$ sequential network evaluations, NAR decoding significantly reduces computational complexity to just a single forward pass, where $|V|$ is the number of items or nodes in the problem.

During **training**, the cNC model uses a bi-objective reward function that considers a linear combination of quality and diversity rewards weighted by the condition weight ω used as input:

$$R_{\text{cNC}}(\mathbf{s}'_\mathbf{i}) = (1 - \omega) \cdot f(\mathbf{s}'_\mathbf{i}) + \omega \cdot \frac{1}{|\mathcal{K}|} \sum_{\mathbf{m} \in \mathcal{K}} d(\mathbf{s}, \mathbf{m}). \tag{3}$$

Here $f(\mathbf{s}'_\mathbf{i})$ is the objective value of the generated solution s'_i, and $d(\mathbf{s}'_\mathbf{i}, \mathbf{m})$ is a distance metric measuring the dissimilarity between the new solution and the reference solution $\mathbf{m}$ [1].

Rather than uniformly sampling the exploration weight ω during different training episodes, we observed that focusing on extreme values (near 0 and 1) facilitates more effective learning of the Pareto front boundaries. Consequently, in each episode we sample ω from a Beta distribution with parameters $\alpha = \beta = 0.2$, which biases sampling towards these extreme values. The improved Pareto front achieved by the cNC model using this biased sampling is empirically validated in Appendix D.

During inference, we leverage the cNC model's ability to adjust the exploration weight using a cooling schedule based on the remaining execution budget. Specifically, for each iteration t out of a

[1] We use the bitwise Hamming distance for binary solution vectors.

total budget $T_{\max}$, the exploration weight is defined as:

$$\omega(t) = \omega_{\text{start}} \left(1 - \frac{t}{T}\right)^{\phi} \tag{4}$$

where ω_{start} is the initial exploration weight and $\phi > 0$ is a cooling factor that controls the curvature of the cooling schedule. We analyze the effect of different cooling schedules in the Appendix D.2.

4.3 Router

The Router dynamically assigns agents to either the Improvement Module or the Exploration Module based on their performance.

By default, agents use the Improvement Module to refine their current solutions. To prevent stagnation, the router monitors the number of consecutive iterations during which an agent fails to achieve an improvement in its solution. If this count exceeds a predefined patience threshold (N_{patience}), the Router redirects the agent to the Exploration Module to start the search from a new candidate solution. We provide a detailed ablation study in Appendix E that systematically varies N_{patience} and analyzes its impact on both solution quality and convergence.

To provide a comprehensive understanding of the interactions between the modules and the router, we present the pseudo-code for MALOTE's inference procedure in Appendix F.

4.4 Neural Network Architecture

Both the improvement policy π_{NI} and the exploration policy π_{cNC} are parameterized by Graph Transformers (GT) [Dwivedi and Bresson, 2020], which ensure size- and permutation-invariance through self-attention over graph-structured inputs. GT encodes node and edge features into latent embeddings, which are decoded into policy-specific logits. For π_{NI}, the decoder outputs one logit per node in V, indicating the probability of flipping each corresponding node. For π_{cNC}, it produces a $|V| \times 2$ heatmap, assigning scores for class 0 or 1, which is greedily decoded into a solution. Detailed hyperparameters and training configurations of the GT model are presented in Appendix G.

5 Experiments

We evaluate MALOTE on two benchmark problems, Maximum Cut (MC) [Dunning et al., 2018] and Maximum Independent Set (MIS) [Lawler et al., 1980], with formal definitions and feature processing details provided in Appendix A.

For each problem, we train the NI and cNC models on randomly generated Erdős–Rényi (ER) graphs with a 15% edge probability and sizes ranging from 50 to 200 nodes. During inference, we use a population size of $|A| = 20$ per instance, set a patience threshold of $N_{\text{patience}} = 500$ steps for the router, and apply a linear cooling factor $\phi = 1$ for the exploration weight of the cNC model. Additional parameters used for inference are provided in Appendix G.

Following recent studies [Ahn et al., 2020, Böther et al., 2021, Zhang et al., 2023], we assess MALOTE's generalization on larger ER graphs (700–800 nodes) and on more challenging RB graphs [Xu and Li, 2000] (800–1200 nodes).

For a more comprehensive comparison, we include exact methods using the GUROBI solver [Gurobi Optimization, LLC, 2023], greedy heuristics, metaheuristics such as Genetic Algorithms (GA) [Kramer and Kramer, 2017] and Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], specialized algorithms like BURER [Burer et al., 2002] for MC and K$_A$MIS [Lamm et al., 2016] for MIS, and learning-based methods including S2V-DQN [Khalil et al., 2017], ECO-DQN [Barrett et al., 2020], FlowNet [Zhang et al., 2023], ANYCSP [Tönshoff et al., 2023], MARCO [Garmendia et al., 2024] for MC, and additionally DGL [Böther et al., 2021], LwD [Ahn et al., 2020], INTEL [Li et al., 2018] and DiffUCO [Sanokowski et al., 2024] for MIS. Together, these methods span a broad spectrum of algorithms, including the state-of-the-art techniques. For full details on the benchmark methods, see Appendix H.

Exact methods, heuristics, and metaheuristics have been executed using a cluster with 32 *Intel Xeon X5650* CPUs. MALOTE and the other learning-based methods have been implemented using *PyTorch 2.0*, and a *Nvidia H100* GPU has been used to train the models and perform inference [2].

Table 1: MC and MIS performance table. The best overall results are highlighted in bold. *Used to compute the ratios.

			ER700-800			RB800-1200		
	Method	**Type**	**Objective ↑**	**Ratio ↑**	**Time ↓**	**Objective ↑**	**Ratio ↑**	**Time ↓**
Maximum Cut (MC)	GUROBI	Exact	24048.93	0.992	10m	23729.44	0.747	10m
	Greedy	Heuristic	23774.79	0.980	0.03s	30619.32	0.964	0.04s
	GA	Metaheuristic	24211.64	0.999	1m	31762.89	1.000	1m
	PSO	Metaheuristic	24201.78	0.999	1m	31764.80*	1.000	1m
	BURER	Specialized	24235.93*	1.000	1m	29791.52	0.938	1m
	S2V-DQN	RL / NC	21581.79	0.890	10.1s	22014.93	0.693	13.4s
	FlowNet	UL / NC	21727.19	0.896	2.25m	23410.60	0.785	2.97m
	ECO-DQN	RL / NI	24114.06	0.994	2.10m	29638.78	0.933	3.00m
	ANYCSP	RL / NI	24211.00	0.999	35.7m	31544.76	0.993	42.4m
	MARCO	RL / NI	24205.97	0.998	1.67m	29780.71	0.938	2.75m
	cNC	RL / NC	23394.25	0.965	0.01s	18405.78	0.579	0.01s
	MALOTE$_s$	RL / NI + cNC	24231.19	1.000	30s	31766.79	1.000	30s
	MALOTE	RL / NI + cNC	**24238.82**	**1.000**	5m	**31767.55**	**1.000**	5m
Maximum Independent Set (MIS)	GUROBI	Exact	43.64	0.970	10m	41.34	0.957	10m
	Greedy	Heuristic	38.85	0.863	0.05s	37.78	0.875	0.06s
	GA	Metaheuristic	43.97	0.977	1m	41.39	0.959	1m
	PSO	Metaheuristic	43.69	0.971	1m	41.19	0.955	1m
	KAMIS	Specialized	44.98*	1.000	1m	**43.15***	**1.000**	1m
	DGL	SL / Hybrid	38.71	0.861	11s	32.32	0.750	2.61s
	INTEL	SL / Hybrid	41.13	0.913	10s	34.24	0.794	2.44s
	LwD	RL / NI	41.17	0.915	4s	34.50	0.799	0.86s
	FlowNet	UL / NC	41.14	0.914	2s	37.48	0.868	0.46s
	DiffUCO	UL / NC	42.21	0.938	2.6s	38.87	0.900	0.42s
	MARCO	RL / NI	43.78	0.973	17s	40.13	0.930	6s
	cNC	RL / NC	38.96	0.866	0.03s	35.39	0.820	0.04s
	MALOTE$_s$	RL / NI + cNC	44.96	1.000	30s	39.97	0.926	30s
	MALOTE	RL / NI + cNC	**45.38**	**1.009**	5m	40.97	0.949	5m

5.1 Performance Experiments

We evaluate our framework using three key metrics: the average objective value $f(\mathbf{s})$ over the entire dataset, the performance ratio $R(\mathbf{s}) = \frac{f(\mathbf{s})}{f_{\text{best}}}$, where f_{best} represents the best known objective value for the instance, and the average execution time per instance.

In this first experiment, we aim to showcase the best possible performance of each method by setting time or iteration limits tailored to their design. Exact methods, such as Gurobi, are allowed 10 minutes to find high-quality solutions. Heuristics run until they reach their final solution. Metaheuristics and specialized algorithms run for 1 minute per each instance. Learning-based baselines follow the limits and setups proposed in their original papers. Our proposed method, MALOTE, is evaluated with a short 30-second run (MALOTE$_s$), and a longer time limit of 5-minutes (MALOTE).

As shown in Table 1, MALOTE outperforms all evaluated methods on the MC-ER, MC-RB, and MIS-ER benchmarks, and achieves the best performance among learning-based approaches across all four benchmarks. Notably, our standalone cNC policy, run with the exploration weight set to 0, achieves reasonable performance in a single inference pass, offering a remarkably fast alternative to traditional greedy heuristics.

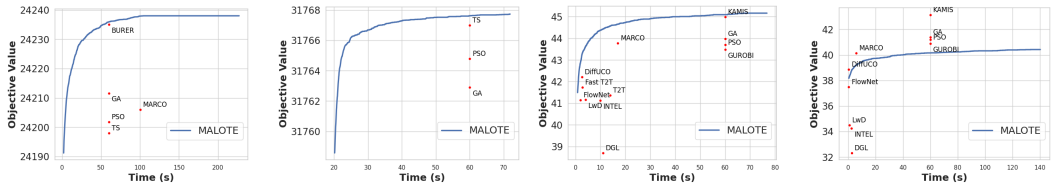
Regarding computational time, it is important to note that CPU vs. GPU runtimes are not directly comparable. While we include average runtime primarily to provide a coarse view of computational cost, our focus is on solution quality and anytime performance. In some experiments, MALOTE is given a larger time budget than certain baselines, mainly because it had not yet converged to its best

[2]The source code, along with all scripts necessary to reproduce the experimental results presented in this paper, will be made publicly available upon manuscript acceptance.

solution. However, as we will demonstrate in the following experiment, MALOTE still outperforms these baselines even when given a substantially shorter execution time.

5.2 Anytime performance

While MALOTE performs strongly under the proposed time limits (5 minutes and 30 seconds), it is also important to assess its behavior under varying time budgets, that is, its anytime performance. To this end, Figure 3 shows the best objective values obtained over time across different benchmarks. The results demonstrate that MALOTE consistently delivers superior solution quality compared to all evaluated baselines across all MC datasets and on ER graphs in the MIS problem.



(a) MC - ER 700-800 (b) MC - RB 800-1200 (c) MIS - ER 700-800 (d) MIS - RB 800-1200

Figure 3: Anytime performance of MALOTE and baselines.

Additional Experiments In Appendix J, we compare MALOTE against equivalent metaheuristics, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), under varying time budgets. The impact of centralized memory during training is analyzed in Appendix I. Lastly, ablation studies on MALOTE, along with their corresponding anytime performance curves, are reported in Appendix J.1.

6 Conclusions

In this paper, we introduced **MALOTE**, a multi-agent reinforcement learning framework that balances exploration and exploitation for robust anytime performance. By combining multiple agents, each with exploration and improvement modules, and a centralized memory, MALOTE fosters diversity and mitigates premature convergence. Experiments on Maximum Cut and Maximum Independent Set confirm its ability to steadily improve solutions with extended computational budgets, outperforming state-of-the-art methods on three of four evaluated benchmarks.

Limitations and Future Work While MALOTE demonstrates promising results, there are numerous opportunities for future investigation. One limitation comes from the information loss incurred when the cNC policy considers only a subset of visited solutions. This sub-sampling discards potentially valuable information about the search history. Future work could explore methods to mitigate this information loss, such as developing techniques to aggregate or summarize the dynamically changing information of past visited solutions, perhaps through learned embeddings or more sophisticated memory structures. Another area for improvement lies in the control flow between the exploration and improvement modules. Currently, the routing strategy that determines when to execute each module is relatively simple. Exploring more intelligent and adaptive routing strategies could further refine performance and potentially reduce unnecessary computation by dynamically routing to each module based on the current state of the search. Finally, it would be particularly interesting to explore the use of the conditioned Neural Constructive policy in multi-objective combinatorial optimization problems, where balancing competing objectives could further highlight its strengths.

References

Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International conference on machine learning*, pages 11–20. PMLR, 2019.

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

Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3243–3250, 2020.

Thomas D Barrett, Christopher WF Parsonson, and Alexandre Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

Julien Baste, Michael R Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022.

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

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

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: a review. *Evolutionary computation*, 25(1):1–54, 2017.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations*, 2021.

Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *arXiv preprint arXiv:2103.03012*, 2021.

Samuel Burer, Renato DC Monteiro, and Yin Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12(2):503–521, 2002.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2): 156–172, 2008.

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

Felix Chalumeau, Refiloe Shabe, Noah De Nicola, Arnu Pretorius, Thomas D Barrett, and Nathan Grinsztajn. Memory-enhanced neural solvers for efficient adaptation in combinatorial optimization. *arXiv preprint arXiv:2406.16424*, 2024.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Paulo Roberto de O da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning*, pages 465–480. PMLR, 2020.

Marco Dorigo and Thomas Stützle. *Ant colony optimization: overview and recent advances*. Springer, 2019.

Iain Dunning, Swati Gupta, and John Silberholz. What works best when? a systematic evaluation of heuristics for max-cut and qubo. *INFORMS Journal on Computing*, 30(3):608–624, 2018.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Florian Felten, El-Ghazali Talbi, and Grégoire Danoy. Multi-objective reinforcement learning based on decomposition: A taxonomy and framework. *Journal of Artificial Intelligence Research*, 79:679–723, 2024.

Andoni I Garmendia, Josu Ceberio, and Alexander Mendiburu. Neural improvement heuristics for graph combinatorial optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Andoni I Garmendia, Quentin Cappart, Josu Ceberio, and Alexander Mendiburu. Marco: a memory-augmented reinforcement framework for combinatorial optimization. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 6931–6939, 2024.

Fred Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

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.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *ECAI 2020*, pages 443–450. IOS Press, 2020.

André Hottung, Mridul Mahajan, and Kevin Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. *arXiv preprint arXiv:2402.14048*, 2024.

Jorik Jooken, Pieter Leyman, Tony Wauters, and Patrick De Causmaecker. Exploring search space trees using an adapted version of monte carlo tree search for combinatorial optimization problems. *Computers & Operations Research*, 150:106070, 2023.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee, 1995.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220 (4598):671–680, 1983.

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

Oliver Kramer and Oliver Kramer. *Genetic algorithms*. Springer, 2017.

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.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F Werneck. Finding near-optimal independent sets at scale. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 138–150. SIAM, 2016.

Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*, 2019.

Fu Luo, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

Sajad Modaresi, Denis Sauré, and Juan Pablo Vielma. Learning in combinatorial optimization: What and how to explore. *Operations Research*, 68(5):1585–1604, 2020.

Ankur Nath and Alan Kuhnle. A benchmark for maximum cut: Towards standardization of the evaluation of learned heuristics for combinatorial optimization. *arXiv preprint arXiv:2406.11897*, 2024.

Shengjie Ren, Zhijia Qiu, Chao Bian, Miqing Li, and Chao Qian. Maintaining diversity provably helps in evolutionary multimodal optimization. *arXiv preprint arXiv:2406.02658*, 2024.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.

Jiwoo Son, Zhikai Zhao, Federico Berto, Chuanbo Hua, Changhyun Kwon, and Jinkyoo Park. Neural combinatorial optimization for real-world routing. *arXiv preprint arXiv:2503.16159*, 2025.

Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In *International Conference on Machine Learning*, pages 32859–32874. PMLR, 2023.

Rui Sun, Zhi Zheng, and Zhenkun Wang. Learning encodings for constructive neural combinatorial optimization needs to regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20803–20811, 2024.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 4280–4288, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. Deepaco: neural-enhanced ant systems for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33: 1621–1632, 2020.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Changliang Zhou, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Instance-conditioned adaptation for large-scale generalization of neural combinatorial optimization. *arXiv preprint arXiv:2405.01906*, 2024.

A Application to the Maximum Cut and Maximum Independent Set

In this study, we address two fundamental graph-based combinatorial optimization problems: Maximum Cut (MC) [Dunning et al., 2018] and Maximum Independent Set (MIS) [Lawler et al., 1980]. Both problems are defined on an undirected graph $G = (V, E)$, where V represents the set of nodes and E denotes the set of edges.

A.1 Maximum Cut (MC)

The Maximum Cut problem seeks to partition the node set V into two disjoint subsets V_1 and V_2 such that the number of edges between these subsets, known as the cut size, is maximized. Formally, given a binary solution vector $\mathbf{s} \in \{0, 1\}^{|V|}$, where $\mathbf{s}_u = 0$ if node u is assigned to V_1 and $\mathbf{s}_u = 1$ if assigned to V_2, the objective function can be expressed as:

$$\max_{\mathbf{s}} \sum_{(u,v)\in E} \delta(\mathbf{s}_u \neq \mathbf{s}_v), \tag{5}$$

where $\delta(\cdot)$ is the Kronecker delta function, which equals 1 if its argument is true and 0 otherwise.

A.2 Maximum Independent Set (MIS)

The Maximum Independent Set problem aims to identify the largest possible subset of nodes $S \subseteq V$ such that no two nodes in S are adjacent; that is, $(u, v) \notin E$ for all $u, v \in S$. Formally, given a binary solution vector $\mathbf{s} \in \{0, 1\}^{|V|}$, where $\mathbf{s}_u = 1$ if node u is included in the independent set and $\mathbf{s}_u = 0$ otherwise, the objective function can be formulated as:

$$\max_{\mathbf{s}} \sum_{u\in V} \mathbf{s}_u \quad \text{subject to} \quad \mathbf{s}_u + \mathbf{s}_v \leq 1 \quad \forall (u, v) \in E. \tag{6}$$

A.3 Feature Encoding

As discussed in Section 4, both the NI and cNC policies rely on node and edge features to iteratively modify or construct a solution, respectively. These features capture information from the problem instance, the current solution (in the case of NI), a set of reference solutions stored in memory, or a conditioning value (used by cNC).

For both MC and MIS, **instance information** is derived from the graph's adjacency matrix (and optionally, edge weights). This structure is encoded as edge features, where each edge is represented by a binary vector using a one-hot encoding scheme to indicate the presence or absence of a connection between node pairs.

In binary problems like MC and MIS, a **solution** can be naturally represented as a binary node feature vector, assigning each node a feature value of 0 or 1. To encode **memory information** in the NI policy, we follow the approach of Garmendia et al. [2024], where the k nearest solutions in memory $\mathcal{M}$, measured by their similarity to the current solution, are selected and aggregated. Similarity is computed by the inner product $\langle \mathbf{s}, \mathbf{s}_i \rangle$ between the current solution $\mathbf{s}$ and each solution $\mathbf{s}_i \in \mathcal{M}$. The resulting aggregation is a weighted average over the k selected solutions, producing a continuous node feature vector of dimension $|V|$, which serves as an additional input to the NI policy.

In the cNC policy, the fixed set of reference solutions $\mathcal{K}$ is encoded by assigning one feature dimension per solution $\mathbf{s}_i \in \mathcal{K}$. Thus, the reference set contributes $|\mathcal{K}|$ additional node features. Additionally, the conditioning parameter, or exploration weight ω, is appended as a constant scalar feature to each node, providing one more node feature dimension.

In terms of feature breakdown:

- **NI policy:** Each edge is assigned one feature to represent the problem instance (e.g., edge presence or weight). Each node receives two features: one from the current solution (a binary value) and another from the aggregated memory information.
- **cNC policy:** Each edge also has one instance-based feature. Each node receives $|\mathcal{K}|$ features from the reference solutions, plus one feature for the exploration weight, resulting in a total of $|\mathcal{K}| + 1$ node features.

A.3.1 Exploiting Symmetries

Exploiting the symmetry in the MC problem, where inverting all node assignments yields solutions with identical objective values, we introduce an specific encoding strategy for the cNC model. We preset the first node to a fixed set (e.g., value = 1) to establish a reference point. Additionally, we assign a learnable parameter to this first node and a separate learnable parameter to the remaining nodes, to allow the model to recognize the reference node.

B Application of MALOTE to Alternative Problems

To adapt MALOTE to a new combinatorial optimization problem, three key components must be addressed:

Input Feature Representation. The first step involves defining suitable node and edge features specific to the problem at hand. These features encode the structure and constraints of the problem instance. Once defined, they are processed by the encoder—such as the Graph Transformer (GT) architecture used in this work (detailed in Appendix G)—to produce node (and optionally edge) embeddings.

Action Mapping. The second step requires adapting how these embeddings are used to take actions. In the case of NI, this means modifying a current solution based on the embeddings. For the cNC policy, this entails constructing new solutions from scratch using the embeddings and the current exploration weight.

Memory and Similarity Integration. Finally, integrating the shared memory component involves defining a suitable representation for storing solutions and designing a problem-specific similarity metric to retrieve the k-nearest neighbors. These metrics are also used to condition the cNC policy during training and inference.

In the following, we illustrate how these three components can be instantiated for three representative problems: the Traveling Salesman Problem (TSP), the Knapsack Problem (KP), and the Job-Shop Scheduling Problem (JSSP).

B.1 Graph Features in Alternative Problems

The formulation of node and edge features varies depending on the problem domain.

In the TSP, node features can be defined using the spatial coordinates of the cities, while edge features typically represent the distances between city pairs.

In the KP, nodes correspond to items, with their primary node feature being the item's weight. The total knapsack capacity can either be appended as a global scalar feature to each node or, alternatively, item weights can be normalized by the knapsack capacity to reflect their relative contribution.

The JSSP presents a more complex structure, as it involves two distinct entities: jobs and machines. A natural representation is to use a heterogeneous graph, where each node corresponds to either a job or a machine. Node features should therefore include an indicator denoting the type of node (job or machine). Edge features are used to encode processing times: an edge from job node i to machine node j carries the processing time required for job i on machine j, if such an operation is defined.

B.2 Actions in Alternative Problems

Defining actions varies significantly depending on the type of policy: NI or cNC, and the nature of the problem.

In TSP and other permutation-based problems, NI actions correspond to modifications of the current permutation of nodes. These include common local operators such as swap, insertion, or 2-opt, all of which can be represented by a pair of nodes (i.e., an edge). Consequently, it is advantageous to use edge embeddings, allowing the decoder to output edge logits. At each step, a probability distribution over edges defines the likelihood of each possible modification.

In the cNC setting for TSP, solutions are typically constructed greedily using edge-based heatmaps, as in non-autoregressive (NAR) decoding frameworks [Joshi et al., 2020], where edges are scored and selected sequentially to form a valid tour.

For the KP, NI actions involve adding or removing individual items from the current solution. In contrast, cNC policies apply node-level heatmaps to construct solutions directly in a single pass, where the selection probabilities are derived from node embeddings.

In the JSSP, actions must respect the precedence and machine constraints that govern feasible schedules. For NI policies, actions correspond to local rescheduling moves, such as swapping the execution order of two operations on the same machine or shifting an operation earlier or later in time within its machine queue. These actions can be encoded as edges between operation nodes on the same machine, and thus, similar to TSP, edge embeddings and logits can be used to select the next local modification.

In cNC, the construction process involves assigning operations to time slots in a way that respects both job precedence and machine availability. A natural approach is to define a greedy NAR decoding process, where the model outputs a heatmap over possible job-machine-time triplets. At each decoding step, the highest-scoring operation is scheduled next, progressively constructing a valid schedule until all operations are placed.

B.3 Similarity Measures

In the TSP, a convenient way to represent visited solutions is by storing the set of edges that belong to those tours, via a one-hot encoding in edge features. Regarding the similarity between solutions, permutation-based distance metrics could be used, such as Kendall Tau, or Cayley distance.

In the KP, solutions are binary vectors indicating whether each item is included (1) or excluded (0) from the knapsack, and thus, can be represented as node features. The binary format of solutions is naturally suited to Hamming distance, which counts the number of differing bits between two solutions.

For the JSSP, there are multiple alternatives: one is to encode solutions as a vector of operation start times, and compute similarity using Euclidean distance or mean absolute error between these vectors. Another option is to represent solutions as a sequence of operation-machine assignments and use sequence-based metrics such as Kendall Tau or edit distance to compare them.

C Training Details of the Improvement Module

This section details the training procedure for the NI model within the improvement module. The training process consists of multiple episodes, each involving a batch of randomly generated problem instances. For each instance in the batch, a set of agents $\mathcal{A}$ is initialized, each agent A_i starting with a candidate solution $\mathbf{s}_i^0$ and a centralized memory $\mathcal{M}$ shared by the agents.

Each episode proceeds through T optimization steps. In each step t, every agent proposes a modification to its current candidate solution based on the problem instance, its current solution, and the shared memory. After applying the proposed modification, rewards R_i^t are computed according to Equations 1 and 2, and the updated solutions are stored in the memory.

At the end of each episode, the parameters of the NI model are updated using the REINFORCE algorithm [Williams, 1992], maximizing the expected cumulative reward across all agents over T steps. The objective can be formally defined as:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\pi}_{\boldsymbol{\theta}}} \left[\sum_{i=1}^{|\mathcal{A}|} \sum_{t=0}^{T-1} \gamma^t R_i^t \right], \tag{7}$$

where $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ denotes the joint policy of all agents, parameterized by $\boldsymbol{\theta}$, $|\mathcal{A}|$ is the number of agents or the population size, and γ is the discount factor that prioritizes immediate rewards over distant ones.

Algorithm 1 NI - Training Procedure

Input: Distribution of instances $\mathcal{D}$, population size $|\mathcal{A}|$, number of training episodes E, episode length T, NI policy π with parameters θ
Training:
Initialize policy network π_θ
for episode $e = 1$ to E **do**
  $\mathcal{I} \leftarrow$ Sample a batch of M problem instances from $\mathcal{D}$
  for each instance m in the batch **do**
    Initialize:
    for each Agent $i \in \mathcal{A}$ **do**
      $s_i^0 \leftarrow \text{RandomInitialization}()$
    end for
    Initialize shared memory $\mathcal{M}_0 \leftarrow \varnothing$
    Main loop:
    for $t = 0$ to $T - 1$ **do**
      for each Agent $i \in \mathcal{A}$ **do**
        $x_i^t \leftarrow (\mathcal{I},\, s_i^t,\, \mathcal{M}_t)$
        $a_i^t \sim \pi_\theta(\cdot \mid x_i^t)$
        $s_i^{t+1} \leftarrow \text{ApplyAction}\big(s_i^t,\, a_i^t\big)$
        $R_i^t \leftarrow R_{\text{obj}} + w_{\text{rep}} \cdot R_{\text{rep}}$
        $\mathcal{M}_{t+1} \leftarrow \text{UpdateMemory}(\mathcal{M}_t,\, s_i^{t+1})$
      end for
    end for
  end for
  Policy Update:
  $b \leftarrow \frac{1}{|\mathcal{A}|} \sum_{i=1}^{|\mathcal{A}|} \sum_{t=1}^{T} R_i^t \quad \{\textit{baseline}\}$
  $A_i^t \leftarrow \left(\sum_{k=0}^{T-t} \gamma^k\, R_i^{t+k}\right) \;-\; b \quad \{\textit{advantage}\}$
  $\Delta\theta \leftarrow \eta \sum_{i=1}^{|\mathcal{A}|} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_i^t \mid x_i^t) \cdot A_i^t.$
  $\theta \leftarrow \theta \;+\; \Delta\theta$
end for
Output: Trained policy parameters θ

To reduce the variance of the gradient estimator, a baseline strategy is usually adopted. Specifically, the average reward across all agents within the episode is used as the baseline:

$$b = \frac{1}{|\mathcal{A}|} \sum_{i=1}^{|\mathcal{A}|} \sum_{t=0}^{T-1} R_i^t. \tag{8}$$

The advantage for each agent i at each step t is then computed as:

$$A_i^t = \left(\sum_{k=0}^{T-t-1} \gamma^k R_i^{t+k} \right) - b. \tag{9}$$

The policy parameters are updated using the REINFORCE gradient:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{i=1}^{|\mathcal{A}|} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_i^t \mid x_i^t) \cdot A_i^t. \tag{10}$$

The complete training procedure is outlined in Algorithm 1. For clarity, we omit the batch size summation in the parameter update calculations within the algorithm.

C.1 Training Hyperparameters

Table 2 presents the training hyperparameters used. To promote generalization to different graph sizes, we vary the size of the generated instances across episodes. Specifically, the number of nodes for each instance is sampled uniformly in the range $[50, 200]$.

Table 2: Used Training Hyperparameters.

Hyperparameter	**Value**
Episodes	100k
Min Problem Size	50
Max Problem Size	200
Batch Size	128
Population Size	5
Episode Length T	3
γ (Discount)	0.95
Optimizer	AdamW
Learning Rate η	5×10^{-5}
Betas (AdamW)	(0.9, 0.95)
Weight Decay (AdamW)	0.1
Memory Type	Marco-shared
Random Seed	42

D Conditioned Neural Constructive

The conditioned Neural Constructive (cNC) offers significant advantages over training multiple independent models for varying exploration preferences. Instead of discretizing the preference space and training a separate model for each discrete value, cNC learns a continuous function conditioned on the exploration weight. This approach has several key benefits: (1) it requires training only a single model; and (2) it allows for fine-grained control over the exploration-exploitation balance by sampling any value within the continuous preference range.

However, training a cNC requires careful consideration of the training process. In each training episode, a batch of random problem instances is initialized along with a set of diverse candidate solutions. These solutions are used as reference to encourage the generation of distinct and high-quality solutions. Additionally, a random exploration weight is sampled from the range $[0, 1]$ in each episode, providing the conditioning signal for the network. This stochastic sampling of the exploration weight is essential for the cNC to learn a robust mapping between exploration preference and constructive behavior.

The cNC policy is trained using the REINFORCE algorithm, following the same procedure as the NI policy described in Appendix C. The main distinction lies in the baseline computation: rather than aggregating rewards from multiple agents, the cNC training performs multiple rollouts (i.e., NAR decoding of solutions) for each instance, and uses the average reward across these rollouts as the baseline.

The general training process is outlined in Algorithm 2. For training the cNC in this work, we used a batch of 64 instances with 500 nodes per episode, $10,000$ episodes, 20 considered solutions ($|\mathcal{K}| = 20$), and 10 rollouts per instance ($N_{roll} = 10$).

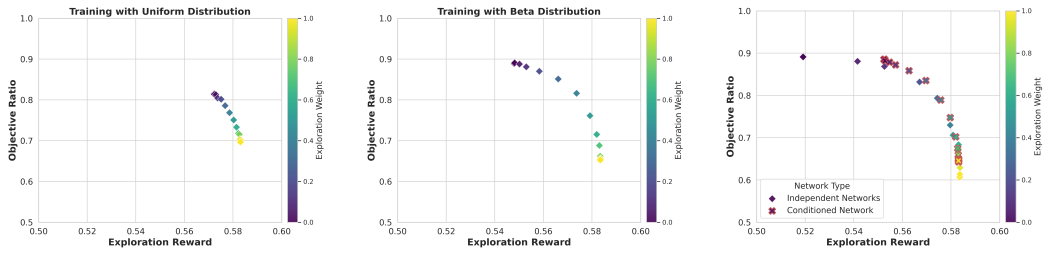
D.1 Sampling of the Exploration Weight during Training

Initially, we experimented with uniform sampling of the exploration weight. However, as shown in Figure 4a, this approach failed to adequately cover the Pareto front. To address this, we employed a Beta distribution with parameters $\alpha = \beta = 0.2$. This U-shaped distribution emphasizes the sampling of extreme exploration weights (close to 0 and 1), which proved crucial for the cNC to effectively learn the boundaries of the Pareto front (Figure 4b).

Furthermore, we conducted a comparative study against a configuration using multiple independently trained networks. In this baseline, we discretized the exploration preference into 11 values (0.0, 0.1, ..., 1.0) and trained a separate model for each. This required training 11 distinct networks. The results presented in Figure 4c demonstrate that the cNC is able to provide solutions that slightly dominate those obtained by independently trained networks.

Algorithm 2 cNC - Training Procedure

Input: Distribution of instances $\mathcal{D}$, number of training episodes E, number of rollouts N_{roll}, number of considered solutions $|\mathcal{K}|$, cNC policy π with parameters θ, batch size B
Training:
Initialize policy network π_θ
for episode $e = 1$ to E **do**
  $\mathcal{I} \leftarrow$ Sample a batch of B problem instances from $\mathcal{D}$
  $\mathcal{K} \leftarrow$ Initialize $|\mathcal{K}|$ random solutions
  $\omega \leftarrow$ Sample random exploration weight $\in [0, 1]$
  for each instance i in the batch **do**
    $x_i \leftarrow (\mathcal{I}, \mathcal{K}, \omega)$ {Define input state}
    $l_i \leftarrow \pi_\theta(x_i)$ {Compute logits}
    $p_i \leftarrow \text{Softmax}(l_i)$ {Compute probabilities}
    for $r = 1$ to N_{roll} **do**
      $s_i^r \leftarrow \text{Rollout}(p_i)$ {Sample a solution}
      $R^i_{\text{obj}} \leftarrow f(s_i^r)$
      $R^i_{\text{dist}} \leftarrow \text{ComputeDistance}(s_i^r, \mathcal{K})$
      $R_i^r \leftarrow (1-\omega) \cdot R^i_{\text{obj}} + \omega \cdot R^i_{\text{dist}}$
    end for
  end for
  Policy Update:
  $b_i \leftarrow \frac{1}{N_{\text{roll}}} \sum_{r=1}^{N_{\text{roll}}} R_i^r$ {*per-instance baseline*}
  $A_i^r \leftarrow R_i^r - b_i$ {*advantage*}
  $\theta \leftarrow \theta + \eta \sum_{i=1}^{B} \sum_{r=1}^{N_{\text{roll}}} \nabla_\theta \big(\log \pi_\theta(s_i^r \mid x_i)\big) A_i^r$
end for
Output: Trained policy parameters θ



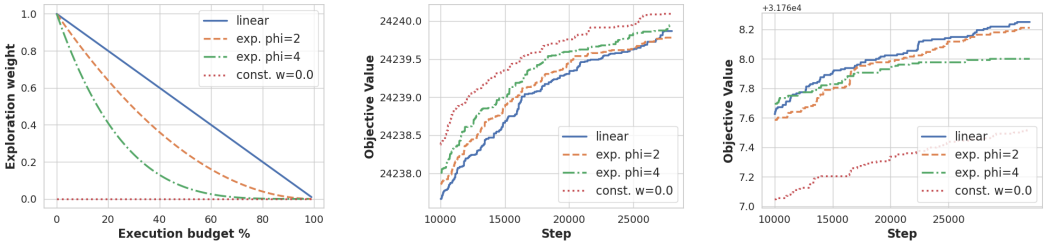
(a) Uniform sampling. (b) Beta distribution. (c) Beta dist. vs. Independent.

Figure 4: Bi-objective performance of different strategies. The plots show the objective value ratio with respect to the best known value (y-axis), against the exploration reward, computed as the average distance to the considered solutions (x-axis), for various exploration weights (colored points). Results are obtained on 100 ER graphs with 100 nodes for the MC problem. (a) cNC trained with uniform sampling of the exploration weight. (b) cNC trained with Beta distribution sampling ($\alpha = \beta = 0.2$). (c) Comparison between the Beta-trained cNC and multiple independently trained networks, each optimized for a specific, discretized exploration weight.

D.2 Cooling Schedule

Once trained the cNC, it permits to dynamically adjust the exploration weight throughout the optimization process. In the initial phases of the optimization process, a strong emphasis on exploration allows the algorithm to investigate a diverse set of solutions. As the process progresses, it becomes advantageous to gradually shift the focus towards exploitation. This transition can be seen as a cooling schedule, analogous to the temperature reduction in simulated annealing [Kirkpatrick et al., 1983].

Equation 4 presents the general formula for the cooling schedule, which dynamically adjusts the weight assigned to exploration based on the budget left. We investigated several cooling strategies,



(a) Different cooling schedules. (b) Avg. obj. curves for ER700-800 (c) Avg obj. curves for RB800-1200

Figure 5: Different cooling schedules for the exploration weight and their performance in ER700-800 and RB800-1200 evaluation datasets for the MC.

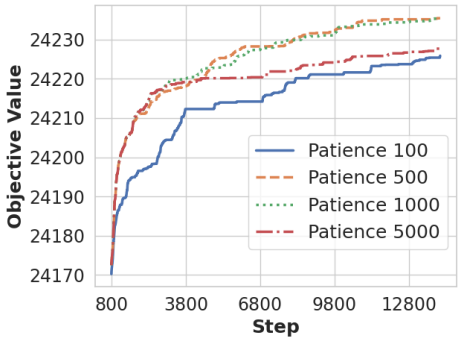


Figure 6: Influence of the Patience Parameter (N_{patience}) on Optimization Performance. The plot shows the evolution of the best objective value found during the optimization process for different values of N_{patience} (100, 500, 1000, and 5000) averaged over 128 ER700-800 graphs for the Maximum Cut problem.

modifying the cooling factor ϕ. We compare the linear cooling used in the paper ($\phi = 1$), with two exponential schedules ($\phi = 2$ and $\phi = 4$), where the exponential decrease of the weight should allow for a faster reduction in exploration compared to linear cooling, and a constant weight ($\omega = 0$) meaning that the policy only focuses on exploitation.

Figure 5a visualizes the cooling curves for each proposed schedule. We evaluated their performance on two distinct MC datasets: ER700-800 (Figure 5b) and RB800-1200 (Figure 5c).

Our results reveal a strong relationship between the optimal level of exploration and whether the evaluation data are drawn from the training distribution (ER) or from a different distribution (RB). For example, the strategy focused solely on exploitation ($\omega = 0$) performs best on the in-distribution ER graphs but exhibits the worst performance on the out-of-distribution RB graphs. This trend extends to the other cooling schedules. Higher exploration rates lead to improved performance on the out-of-distribution RB graphs, while lower exploration rates (emphasizing exploitation) yield better results on the in-distribution ER graphs. These findings underscore the importance of exploration for generalization to unseen data.

E Router Patience

The patience parameter (N_{patience}) is crucial for the router's ability to balance exploitation and exploration in agent behavior. It specifies the number of consecutive iterations in which an agent can fail to improve its solution before being redirected to the Exploration Module to restart the search. Selecting an appropriate N_{patience} involves a trade-off: a low value can cause premature exploration, potentially missing out on good solutions, while a high value can result in excessive computation within a local region of the search space without yielding significant improvements.

Algorithm 3 MALOTE Inference Procedure

1: **Input:** Instance to be solved $\mathcal{I}$, trained NI and cNC policies, population size $|A|$, max steps T_{max}, patience threshold N_{patience}, initial exploration weight ω_{start}, cooling factor ϕ,
2: **Initialize:**
3: **for** each agent $i \in \mathcal{A}$ **do**
4: $s_i^0 \leftarrow \text{RandomInitialization}(\mathcal{I})$, $c_i \leftarrow 0$ {Consecutive non-improving steps}
5: **end for**
6: $\mathcal{M}_0 \leftarrow \varnothing$ {Shared memory}, $s_{\text{best}} \leftarrow \varnothing$ {Stores the best solution found so far}
7: **for** $t \leftarrow 1$ to T_{max} **do**
8: **for** each agent $i \in \mathcal{A}$ **do**
9: **if** $c_i \geq N_{\text{patience}}$ **then**
10: $\omega \leftarrow \omega_{\text{start}} \left(1 - \frac{t}{T_{\text{max}}}\right)$ {Update exploration weight using cooling schedule}
11: $\mathcal{K}_t \leftarrow \text{SelectSubset}(\mathcal{M}_t)$ {Select a subset of solutions from memory for exploration}
12: $s_i^{t+1} \leftarrow \pi_{\text{cNC}}(\mathcal{I}, \mathcal{K}_t, \omega)$ {New solution via Exploration Module}
13: $c_i \leftarrow 0$ {Reset counter}
14: **else**
15: $s_i^{t+1} \leftarrow \pi_{\text{NI}}(\mathcal{I}, s_i^t, \mathcal{M}_t)$ {Improve solution applying a bit-flip}
16: **if** $\text{IsImproved}(s_i^{t+1}, s_i^t)$ **then**
17: $c_i \leftarrow 0$ {Reset counter}
18: **else**
19: $c_i \leftarrow c_i + 1$ {Increase counter}
20: **end if**
21: **end if**
22: $\mathcal{M}_{t+1} \leftarrow \text{UpdateMemory}(\mathcal{M}_t, s_i^{t+1})$ {Update shared memory with new solution}
23: $s_{\text{best}} \leftarrow \text{SelectBetter}(s_{\text{best}}, s_i^{t+1})$ {Update the best solution found}
24: **end for**
25: **end for**
26: **Output:** s_{best}

To determine suitable values for N_{patience}, we evaluated the performance of our method with several different patience settings: 100, 500, 1000, and 5000; in the ER700-800 graph dataset. Our findings indicate that a small patience value (e.g., $N_{\text{patience}} = 100$) hinders performance by prematurely interrupting the search. Conversely, a large patience value (e.g., $N_{\text{patience}} = 5000$) results in slow convergence, as agents persist in local search for too long. In this experiment, optimal performance was achieved with patience values between 500 and 1000, which yielded similar results. Based on these observations, setting the patience to the number of nodes in the graph instance serves as a simple yet effective rule of thumb. The results of this ablation study are presented in Figure 6.

F Pseudocode for MALOTE Inference

Algorithm 3 presents the inference procedure of MALOTE, as described in Section 4. Lines 2-8 detail the initialization of the agents, memory structures, and other key components. The main execution loop, spanning lines 9-27, governs the core interaction and learning process.

Although the pseudo-code illustrates an inner loop that processes each agent sequentially, the actual implementation executes these iterations in parallel.

G Neural Network Architecture

In this section, we detail the neural network architecture employed in our methodology.

Although the MALOTE framework is designed to be modular and compatible with various encoder architectures, including Graph Neural Networks (GNNs) and transformer-based models that embed node and edge features into a latent space, we adopt the Graph Transformer (GT) architecture [Dwivedi and Bresson, 2020] due to its strong empirical performance across a range of combinatorial optimization problems.

GTs extend the standard transformer model [Vaswani et al., 2017] to operate directly on graph-structured data. Unlike traditional transformers, which are tailored for sequential inputs, GTs

incorporate structural information from the graph by integrating edge features, typically derived from the adjacency matrix, into their attention computations. In our implementation, GTs are combined with a Feed-Forward Neural Network (FFNN) decoder to parameterize both the NI and cNC policies.

G.1 Initial Projection

First, we extract the node and edge features as detailed in Appendix A for the problems addressed, and in Appendix B for additional problems. These features are then projected into a shared embedding space of dimension d to initialize the representations:

$$\mathbf{h}^{(0)} = \mathbf{W}_{\text{node}}\mathbf{x} + \mathbf{b}_{\text{node}}, \tag{11}$$

$$\mathbf{e}^{(0)} = \mathbf{W}_{\text{edge}}\mathbf{y} + \mathbf{b}_{\text{edge}}, \tag{12}$$

where $\mathbf{x} \in \mathbb{R}^{d_{\text{node}}}$ and $\mathbf{y} \in \mathbb{R}^{d_{\text{edge}}}$ are the node and edge feature vectors, respectively; $\mathbf{W}_{\text{node}} \in \mathbb{R}^{d \times d_{\text{node}}}$ and $\mathbf{W}_{\text{edge}} \in \mathbb{R}^{d \times d_{\text{edge}}}$ are learnable weight matrices; $\mathbf{b}_{\text{node}} \in \mathbb{R}^d$ and $\mathbf{b}_{\text{edge}} \in \mathbb{R}^d$ are bias vectors; and $\mathbf{h}^{(0)} \in \mathbb{R}^d$ and $\mathbf{e}^{(0)} \in \mathbb{R}^d$ denote the initial node and edge embeddings.

G.2 Graph Transformer Layers

The GT consists of L layers that iteratively refine the node embeddings. Each GT layer comprises two main components: a multi-head self-attention mechanism and a FFNN, both followed by normalization layers.

Multi-Head Self-Attention. Within each GT layer, node embeddings are transformed into Query (Q), Key (K), and Value (V) tensors for each attention head. Assuming a multi-head attention mechanism with h heads and per-head dimension $d_k = \frac{d}{h}$, the transformations are defined as:

$$Q_i = \mathbf{W}_Q^{(i)}\mathbf{h}, \quad K_i = \mathbf{W}_K^{(i)}\mathbf{h}, \quad V_i = \mathbf{W}_V^{(i)}\mathbf{h}, \tag{13}$$

for each head $i = 1, \ldots, h$, where $\mathbf{W}_Q^{(i)} \in \mathbb{R}^{d_k \times d}$, $\mathbf{W}_K^{(i)} \in \mathbb{R}^{d_k \times d}$, and $\mathbf{W}_V^{(i)} \in \mathbb{R}^{d_k \times d}$ are learnable projection matrices for each head.

The multi-head attention mechanism within a GT layer is computed as follows:

$$\text{Attn}_i(Q_i, K_i, V_i) = \left(\text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}} + \mathbf{E}_i\right) \cdot \mathbf{E}_i\right) V_i, \tag{14}$$

where $\mathbf{E}_i \in \mathbb{R}^{|V| \times |V|}$ integrates edge features into the attention scores for head i. Specifically, $\mathbf{E}_i$ is computed by applying a learnable transformation to the edge embeddings:

$$\mathbf{E}_i = \mathbf{W}_e^{(i)}\mathbf{e}^{(0)}, \tag{15}$$

where $\mathbf{W}_e^{(i)} \in \mathbb{R}^{|V| \times |V|}$ is a learnable weight matrix that maps edge embeddings to a weight matrix and a bias matrix to be multiplied and added to the attention scores, respectively.

The outputs from all attention heads are concatenated and projected back to the original embedding dimension using a learnable output matrix:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{Attn}_1, \ldots, \text{Attn}_h)\mathbf{W}_O, \tag{16}$$

where $\mathbf{W}_O \in \mathbb{R}^{h \cdot d_k \times d}$ is a learnable weight matrix.

Residual Connection and Normalization. The attention output is combined with the original node embeddings $\mathbf{h}$ via a residual connection, followed by a normalization function:

$$\mathbf{h}' = \text{Norm}\left(\mathbf{h} + \text{MultiHead}(Q, K, V)\right).$$

Feedforward Neural Network. Subsequent to the attention mechanism, each GT layer applies a FFNN:

$$\mathbf{h}'' = \text{FFNN}(\mathbf{h}') = \text{Norm}\left(\text{Act}\left(\mathbf{W}_1\mathbf{h}'\right)\right)\mathbf{W}_2, \tag{17}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{ff}} \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times d_{\text{ff}}}$ are learnable weight matrices, Act is the activation function, and Norm denotes a normalization function.

Final Residual Connection and Normalization. The output of the FFNN is then combined with the previous normalized embeddings $\mathbf{h}'$ through another residual connection, followed by a final normalization step:

$$\mathbf{h}^{(l+1)} = \text{Norm}\left(\mathbf{h}' + \mathbf{h}''\right). \tag{18}$$

G.3 Decoder

The output from the final GT layer is subsequently processed by an additional FFNN to generate action logits:

$$z_i = \mathbf{W}_{\text{decoder}}\mathbf{h}_i^{(L)}, \tag{19}$$

where $\mathbf{W}_{\text{decoder}} \in \mathbb{R}^{1\times d}$ is a learnable weight matrix, that maps each node's embedding to a single scalar value.

These logits z_i are then normalized using a softmax function over all nodes to produce a probability distribution:

$$p_i = \frac{\exp(z_i)}{\sum_{j\in V}\exp(z_j)}, \tag{20}$$

where p_i represents the probability of selecting node i for a given action.

The interpretation of p_i depends on the setting:

- In the **NI** setting, p_i denotes the probability of flipping the state of node i.
- In the **cNC** setting, it corresponds to the probability of assigning node i to the first subset.

If the action space is defined over **edges** rather than nodes, as mentioned in Appendix B for alternative problems, edge embeddings can be constructed by concatenating the final-layer embeddings of the two connected nodes:

$$\mathbf{e}_{ij} = [\mathbf{h}_i^{(L)} \,||\, \mathbf{h}_j^{(L)}], \tag{21}$$

and passed through an edge-wise FFNN to obtain edge-level logits, and probabilities in the same manner.

Table 3: Selected Model Hyperparameters

Model Hyperparameter	**Value**
Number of layers (L)	3
Hidden dimension (d)	64
Number of heads (h)	8
FFNN hidden dimension	256
Activation function	GeLU
Normalization	LayerNorm
Dropout	0%

G.4 Hyperparameter Selection

Table 3 lists the hyperparameters used in our model. These were determined through a comprehensive hyperparameter study, where different NI models were trained for 10,000 episodes on instances ranging from 20 to 40 nodes, and using a batch size of 1024. We explored various hyperparameter configurations and present the performance results on 100 instances with sizes of 20, 60, 100, and 200 nodes in Table 4.

H Implementation Details of Baseline Methods

In this section, we provide a more detailed description of the methods used in the experiments.

Table 4: Performance under variations from the used hyperparameter setting. We report the average objective value obtained when testing the NI model after the last epoch of training on ER graph instances with 20, 60, 100 and 200 nodes for the MC problem.

Setting	**ER20**	**ER60**	**ER100**	**ER200**
Used Config.	25.6	192.7	503.6	1896.9
Number of Layers				
$L = 2$	25.6	192.4	502.1	1856.7
$L = 4$	25.6	192.6	503.2	1881.4
Hidden Dimension				
$d = 128$	25.6	192.6	502.7	1885.6
Normalization				
Instance	25.6	192.7	503.4	1894.6
RMS	25.6	192.6	503.2	1892.3
Dropout				
Dropout = 20%	25.6	192.7	503.2	1887.6

Maximum Cut. For the MC problem, we used the methods implemented in the Max Cut Benchmark [Nath and Kuhnle, 2024], modifying them to incorporate a time limit as a stopping criterion. Specifically, we employed the GUROBI exact solver [Gurobi Optimization, LLC, 2023] which is not able to obtain optimal solutions in the given budget, but provides an approximate solution. We also used constructive heuristics such as Forward Greedy, which starts with an empty solution and iteratively adds the vertex that provides the largest gain in the objective value. We applied metaheuristic techniques including Tabu Search (TS) [Glover, 1990], which maintains a tabu list to avoid revisiting recently explored solutions. The benchmark also includes learning-based methods, including S2V-DQN [Khalil et al., 2017], a NC method guided by a GNN; ECO-DQN [Barrett et al., 2020], a Neural Improvement variant of S2V-DQN; FlowNet [Zhang et al., 2023], a work that samples from the solution space with Generative Flow Networks; and ANYCSP [Tönshoff et al., 2023], a GNN-based search method for any constraint satisfaction problem.

Additionally, we integrated a Genetic Algorithm (GA) [Kramer and Kramer, 2017], a population-based metaheuristic that evolves solutions through selection, crossover, and mutation operations; Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], a swarm-based optimization technique where a population of candidate solutions, called particles, moves through the solution space guided by individual and collective experiences; BURER [Burer et al., 2002], a specialized algorithm for MC that leverages semidefinite programming relaxations to approximate solutions; and MARCO [Garmendia et al., 2024], a memory-based NCO method that uses a shared memory to guide the search to unvisited solutions.

Maximum Independent Set We reused several methods also for MIS. We employed the GUROBI exact solver, a greedy constructive heuristic that iteratively adds the node maximizing the gain while ensuring feasibility to the MIS constraints. We also implemented a Genetic Algorithm, a PSO, and used the MIS implementations of FlowNet and MARCO.

Apart from these benchmark methods, we included K_AMIS [Lamm et al., 2016], an evolutionary approach that combines graph kernelization, local search, and graph partitioning techniques to solve the MIS problem. Furthermore, we integrated several learning-based methods: DGL [Böther et al., 2021] and INTEL [Li et al., 2018], which combine a policy learnt by supervised learning with tree search; LwD [Ahn et al., 2020], a scalable reinforcement learning framework that adaptively defers element-wise decisions during solution generation to simplify hard decisions; and DiffUCO [Sanokowski et al., 2024], a diffusion model using unsupervised learning to approximate intractable discrete distributions without requiring training data.

I Advantage of Centralized Training for Memory-Based Methods

The NI model (from the improvement module) in our approach differs from MARCO [Garmendia et al., 2024] in its training paradigm. While MARCO employs a decentralized training scheme in which each search thread maintains a private memory, we utilize a centralized training framework with a shared memory accessible to all agents.

We hypothesize that training with a shared memory increases the complexity of the learning task for the NI model. The model must learn to navigate a more intricate decision-making landscape, avoiding redundant actions not only within its own search trajectory, but also across the trajectories of all other agents sharing the memory.

Table 5: Performance Comparison between decentralized (d) and centralized (c) training in MARCO.

Method	**MC**-objective ↑	**MC**-diversity ↑	**MIS**-objective ↑	**MIS**-diversity ↑
MARCO-d	24205.97	0.54	43.78	0.51
MARCO-c	**24221.72**	**0.59**	**44.48**	**0.57**

To isolate the impact of the centralized memory architecture on performance, we conducted a controlled experiment. We trained a MARCO model using the original training hyperparameters reported in [Garmendia et al., 2024], but modified the training process to incorporate a centralized, shared memory, mirroring our approach. We then evaluated both the original decentralized MARCO model (MARCO-d) and this centralized MARCO variant (MARCO-c) on the ER700-800 dataset for both the MC and MIS problems.

Table 5 presents the results, comparing the average objective value and a measure of solution diversity. Diversity is quantified using the average pairwise Hamming distance between the proposed solutions. The results demonstrate that the centralized variant of MARCO, trained with the approach presented in this paper, achieves significantly improved performance in terms of both the objective value and the diversity of solutions, confirming that centralized memory is a key factor driving the performance gains observed in our approach.

J MALOTE as a Neural Population-Based Metaheuristic

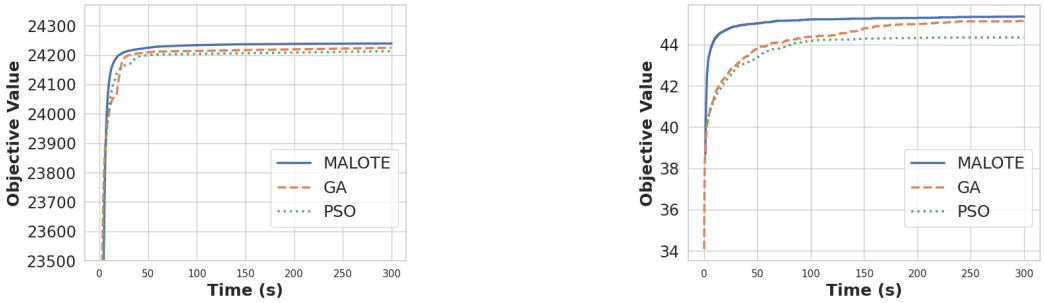
MALOTE shares several conceptual similarities with classical population-based metaheuristics such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). Like these methods, MALOTE maintains a population of candidate solutions and iteratively improves them through interactions and information sharing. In GA, this occurs via selection, crossover, and mutation; in PSO, particles update their positions based on personal and global bests; in MALOTE, the population is updated through neural policies conditioned on a centralized memory that captures the collective search history. The key difference lies in how the update mechanisms are implemented: while GA and PSO use hand-designed heuristics, MALOTE learns its update strategy end-to-end via reinforcement learning, enabling adaptation to problem-specific structures. Additionally, MALOTE leverages neural networks to encode graph-structured inputs and guide exploration, offering a more expressive and learnable framework compared to the fixed dynamics of classical metaheuristics.

To highlight these differences in practice, we compare the anytime performance of MALOTE with that of GA and PSO in Figure 7. This comparison evaluates how solution quality evolves over time under equivalent computational budgets. The results demonstrate that MALOTE outperforms classical metaheuristics having faster convergence, and converging in a higher quality solutions.

These findings support the view that data-driven learning can outperform traditional.

J.1 Ablation Study

To better understand the contribution of key components in the MALOTE architecture, we conduct an ablation study focusing on the Exploration Module and the conditioned Neural Constructive (cNC) policy. Specifically, we evaluate two simplified variants of MALOTE: (1) one without the Exploration Module (denoted as MALOTE - $\{\text{EM}\}$), and (2) one where the Exploration Module is retained, but



(a) MC - ER 700-800 (b) MIS - ER 700-800

Figure 7: Anytime performance of MALOTE and population-based metaheuristics GA and PSO on MC and MIS problems. Each subplot shows the evolution of the best objective value found during the optimization process. Results are shown for both ER700-800 and RB800-1200 graph datasets.

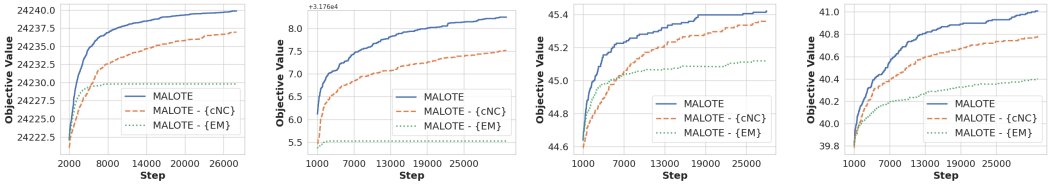
the cNC policy is replaced with a random initialization strategy (denoted as MALOTE - $\{$cNC$\}$). We compare these configurations against the full MALOTE model, limiting each run to a fixed budget of $40|V|$ inference steps, where $|V|$ is the number of nodes in the instance.

Table 6 reports the final objective values obtained, their ratios relative to the best baselines shown in Table 1, and the average runtime required to complete $40|V|$ steps. The results confirm that both the Exploration Module and the cNC policy are critical for achieving strong optimization performance.

Furthermore, Figure 8 illustrates the anytime performance curves of MALOTE and its ablated variants. The results highlight the importance of the Exploration Module in continuously discovering high-quality solutions, and show that incorporating the cNC policy leads to better outcomes than relying on random sampling when the search begins to stagnate.

Table 6: Performance comparison on MC and MIS tasks for different ablation settings. The best overall results are highlighted in bold.

			ER700-800			RB800-1200		
	Method	**Type**	**Objective ↑**	**Ratio ↑**	**Time ↓**	**Objective ↑**	**Ratio ↑**	**Time ↓**
MC	MALOTE - {EM}	RL / NI	24230.26	1.000	1.06m	31765.68	1.000	2.52m
	MALOTE - {cNC}	RL / NI	24237.23	1.000	3.46m	31767.60	1.000	6.10m
	MALOTE	RL / NI + cNC	**24240.16**	**1.000**	7.14m	**31768.03**	**1.000**	9.71m
MIS	MALOTE - {EM}	RL / NI	45.12	1.003	1.00m	40.40	0.936	2.10m
	MALOTE - {cNC}	RL / NI	45.29	1.006	5.31m	40.75	0.944	8.37m
	MALOTE	RL / NI + cNC	**45.44**	**1.010**	5.82m	**41.04**	**0.951**	8.94m



(a) MC - ER 700-800 (b) MC - RB 800-1200 (c) MIS - ER 700-800 (d) MIS - RB 800-1200

Figure 8: Anytime performance of MALOTE and ablations on MC and MIS problems. Each subplot shows the evolution of the best objective value found during the optimization process. Results are shown for both ER700-800 and RB800-1200 graph datasets.

NeurIPS Paper Checklist

1. **Claims**

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: The claims made are based on the experimental results.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Limitations and future work are discussed.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We are open sourcing the code to reproduce the experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We are making public the source code of MALOTE.

   Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We carefully detail all the parameters in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide information regarding the intra-instance variability of non-deterministic methods, and inter-instance variability of all the methods.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.

- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the hardware and execution time necessary to execute the experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The NeurIPS Code of Ethics has been reviewed, and the paper conforms it in every aspect.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: The paper does not use existing assets from other authors.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The research does not involve LLMs

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.