UNIVERSITY OF CALIFORNIA SAN DIEGO

Compositional Value-based Methods for Planning and Generative Modeling

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Chenning Yu

Committee in charge:

Professor Sicun Gao, Chair
Professor Henrik Christensen
Professor Sylvia Herbert
Professor Michael Yip

2025

The dissertation of Chenning Yu is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2025

DEDICATION

To my beloved family, with deepest thanks for always being there.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am deeply thankful for many people who have supported me along the way. Without their guidance and encouragement, this dissertation would not have been possible.

I would like to thank my advisor Professor Sean/Sicun Gao for always supporting and believing in me during this journey. When I started my research journey, I was a student who was merely good at taking exams. Sean was the first person who believed that I could be a good researcher. He encouraged me to explore research topics that I was interested in, and he also helped me to find my research direction. Working with him, I have learned far more than just critical thinking and research acumen - I have also learned the importance of kindness and integrity in life. I am deeply grateful for his unwavering support and guidance throughout my PhD journey.

I thank Ya-Chien Chang, Milan Ganai, Chiaki Hirayama, Zhizhen Qin, Tao Wang, Eric Yu, Hongzhan Yu, Dr. Yaoguang Zhai, Ruipeng Zhang for being my lab mates, collaborators, and my friends. I am grateful for all the memories we created together.

I thank my collaborators, Dr. Jingkai Chen, Professor Chuchu Fan, Professor Sylvia Herbert, Professor Qingbiao Li, Professor Amanda Prorok, Zheng Gong, Mingxin Yu, Luobin Wang, Professor Henrik Christensen, for their support and collaboration. The projects we worked on are not only fruitful but also enjoyable.

I thank Professor Michael Yip for his invaluable contribution as a committee member. His feedback and suggestions have been very helpful in improving my dissertation.

I thank Professor Yian Ma, Professor Lily Weng, Oswin So, Songyuan Zhang for the inspiring academic discussions.

Above all, I am deeply thankful to my family for their constant support and love. Their encouragement gave me the strength to pursue my dreams. I am especially grateful to my wife Wang, my parents Shi and Yu — no words can fully capture my appreciation. I dedicate this dissertation to them.

Chapter 2, in full, is a reprint of Chenning Yu, Sicun Gao, “Reducing Collision Checking in Sampling-based Motion Planning”, NeurIPS, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of Chenning Yu, Qingbiao Li (co-first authors), Sicun Gao, Amanda Prorok, “Accelerating Multi-Agent Planning Using Graph Transformers with Bounded Suboptimality”, ICRA, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in full, is a reprint of Chenning Yu, Hongzhan Yu, Sicun Gao, “Learning Control Admissibility Models with Graph Neural Networks for Multi-Agent Navigation”, CoRL, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, has been submitted for publication of the material as it may appear in Chenning Yu, Sicun Gao, “Improving Compositional Generation with Diffusion Models Using Lift Scores”, ICML, 2025. The dissertation author was the primary investigator and author of this paper.

VITA

2019	B. S. in Computer Engineering, Nanjing University, Nanjing
2021	M. S. in Computer Science, University of California San Diego
2025	Ph. D. in Computer Science, University of California San Diego

PUBLICATIONS

Chenning Yu, Sicun Gao, "Improving Compositional Generation with Diffusion Models Using Lift Scores", *ICML, 2025*

Luobin Wang, Hongzhan Yu, Chenning Yu, Sicun Gao, Henrik Christensen, "Controllable Motion Generation via Diffusion Modal Coupling", *arXiv, 2025*

Mingxin Yu, Chenning Yu, Mohammad Mahdi Naddaf Shargh, Devesh Upadhyay, Sicun Gao, Chuchu Fan, "Efficient Motion Planning for Manipulators with Barrier-Induced Neural Controller", *ICRA, 2024*

Jiasheng Guo, Chenning Yu, Kenan Li, Yijian Zhang, Guoqiang Wang, Shuhua Li, Hao Dong, "Retrosynthesis Zero: Self-improving Global Synthesis Planning Using Reinforcement Learning", *JCTC, 2024*

Milan Ganai, Zheng Gong, Chenning Yu, Sylvia Herbert, Sicun Gao, "Iterative Reachability Estimation for Safe Reinforcement Learning", *NeurIPS, 2023*

Chenning Yu, Qingbiao Li (co-first authors), Sicun Gao, Amanda Prorok, "Accelerating Multi-Agent Planning Using Graph Transformers with Bounded Suboptimality", *ICRA, 2023*

Hongzhan Yu, Chiaki Hirayama, Chenning Yu, Sylvia Herbert, Sicun Gao, "Sequential Neural Barriers for Scalable Dynamic Obstacle Avoidance", *IROS, 2023*, **Robocup Best Paper Award**

Chenning Yu, Hongzhan Yu, Sicun Gao, "Learning Control Admissibility Models with Graph Neural Networks for Multi-Agent Navigation", *CoRL, 2022*

Ruipeng Zhang, Chenning Yu, Jingkai Chen, Chuchu Fan, Sicun Gao, "Learning-based Motion Planning in Dynamic Environments Using GNNs and Temporal Encoding", *NeurIPS, 2022*

Chenning Yu, Sicun Gao, "Reducing Collision Checking for Sampling-Based Motion Planning Using Graph Neural Networks", *NeurIPS, 2021*

ABSTRACT OF THE DISSERTATION

Compositional Value-based Methods for Planning and Generative Modeling

by

Chenning Yu

Doctor of Philosophy in Computer Science

University of California San Diego, 2025

Professor Sicun Gao, Chair

Learning-based models often struggle to generalize from simple training scenarios to more complex tasks. For instance, a model that effectively coordinates small groups of robots may fail on larger teams and denser crowds. Similarly, diffusion models capable of generating images from simple prompts often underperform when faced with more complex requirements. This dissertation presents a unifying framework for constructing compositional value landscapes designed to enable scalable and composable decision-making and generation.

We consider compositional generalization problems that involve either maximizing more complex objective functions and satisfying more complex constraints. We consider a diverse set of planning and generation problem, including trajectory planning and control, multi-agent

coordination, and diffusion-based generative modeling for image synthesis. We use a unifying approach of constructing value landscapes or scoring functions that can be naturally composed through graph-based abstractions or logical operations. They then enable generalization from training on simple to complex inference tasks without additional training. We demonstrate the effectiveness of our approach across a range of domains, including motion planning, multi-agent systems, safe navigation, and visual generative modeling.

Chapter 1

Introduction

In 1956, John McCarthy and his colleagues organized the Dartmouth Conference, marking the birth of artificial intelligence (AI) as a field. At the time, AI was defined as "the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" [115].

Retrospectively, it is interesting to observe the gap between this early definition of AI and many of the current state-of-the-art machine learning systems. On today's Oxford English Dictionary, AI is defined as "capacity of computers or other machines to exhibit or simulate intelligent behaviour" - the conjecture to precisely describe the intelligence is no longer required. Modern AI systems, such as Stable Diffusion and ChatGPT [141, 135], often rely on large amounts of data and complex architectures to learn from examples rather than being explicitly programmed with rules or heuristics. From Symbolic AI to data-driven methods, the shift reflects a move from "understanding intelligence" to "engineering systems that behave intelligently".

Generalization is a key factor to drive this transition. McCarthy's 1956 vision assumed that intelligence could be described in symbolic, logical rules—leading to decades of research into rule-based systems and formal reasoning (e.g., expert systems). However, this approach ran into major roadblocks: (1) Combinatorial explosion in complex real-world tasks. (2) Brittleness:

systems failed when encountering novel or ambiguous inputs. (3) Knowledge acquisition bottleneck: it was hard to hand-code the vast domain knowledge needed. Modern machine learning systems mitigates the bottlenecks by learning patterns from large amount of data, and is more effective to generalize from training data to unseen examples.

Yet, it is still crtical to think about whether modern machine learning systems truly avoid the 3 bottlenecks mentioned above. While they can generalize from training data to unseen examples to some extent, they still struggle with **compositional generalization**, i.e., the ability to flexibly combine known components to solve new, more complex problems. This challenge is evident across various domains of artificial intelligence, including planning and generative modeling. For instance, a learned motion planner can learn to navigate simple environments but often fail when extended to high-dimensional configuration spaces or environments with intricate constraints. Similarly, state-of-the-art generative models such as diffusion models can synthesize compelling images from individual object prompts, but their performance degrades when asked to satisfy multi-object and relational constraints within a single generation.

While it is very tempting to attribute these limitations to the lack of sufficient training data and collect more data to train the models, we propose an alternative approach which may be more efficient, and requires potentially much less data. Our key idea is to develop **compositional value-based methods** — the value functions that are naturally composable through logic-based or graph-based structures. By enabling such functions, we can build decision-making or generative systems that generalize from simple instances to complex tasks, without requiring retraining. These value-based functions serve as inductive scaffolds that encode compositionality directly into the learning process, and they are implemented through models such as Graph Neural Networks (GNNs), Graph Transformers, and diffusion-based scoring mechanisms.

One intriguing property is that these funcitons are typically easy to learn and require relatively small amount of data. Furthermore, the composed value functions can generalize to new scenarios with unseen combinations of the components from the data without additional

retraining. This compositional framework fundamentally shifts the paradigm from learning monolithic models that struggle with compositional generalization to learning simple components that can be flexibly combined, which is both data efficient and effective to generalize.

We investigate this framework through four applications, each targeting a major research area where compositional reasoning is critical. First, we develop GNN-based models for sampling-based motion planning, focusing on reducing collision checking when taking unseen combinations of graph nodes. Second, we propose a graph transformer architecture for bounded-suboptimal multi-agent planning, which leverages transformer-based heuristics to accelerate coordination across large agent teams given solely data from small agent swarm. Third, we introduce control admissibility models using GNNs to enforce safety constraints in multi-agent navigation and demonstrate generalization to unseen combination of more agents. Finally, we extend the compositional framework to the domain of diffusion-based generative modeling, where we design a rejection-sampling strategy guided by compositional lift scores that improves the controllability for combinations of interested objects in text-to-image tasks.

The remainder of this dissertation is organized as follows:

Chapter 2 introduces a GNN-based framework for reducing collision checking in sampling-based motion planning, showcasing how graph-based value functions can accelerate planning in high-dimensional continuous spaces and generalize to unseen combinations of graph nodes.

Chapter 3 presents a graph transformer approach for multi-agent pathfinding with bounded suboptimality, enabling near-optimal planning in environments with large agent populations that is 2-3x than the trained agent population size.

Chapter 4 focuses on learning control admissibility models using GNNs for safe multi-agent navigation, highlighting how graph decomposition and neural estimation support the safety enforcement and generalization to unseen combinations of more agents that is 300x than the trained agent population size.

Chapter 5 explores compositionality in generative modeling, proposing lift-score-based

rejection sampling methods that improve the generalization of controllability to rare combinations of interested objects in text-to-image tasks.

Finally, Chapter 6 concludes with a discussion on the future directions.

Chapter 2

Reducing Collision Checking for Sampling-Based Motion Planning Using Graph Neural Networks

Sampling-based motion planning is a popular approach in robotics for finding paths in continuous configuration spaces. Checking collision with obstacles is the major computational bottleneck in this process. We propose new learning-based methods for reducing collision checking to accelerate motion planning by training graph neural networks (GNNs) that perform path exploration and path smoothing. Given random geometric graphs (RGGs) generated from batch sampling, the path exploration component iteratively predicts collision-free edges to prioritize their exploration. The path smoothing component then optimizes paths obtained from the exploration stage. The methods benefit from the ability of GNNs of capturing geometric patterns from RGGs through batch sampling and generalize better to unseen environments. Experimental results show that the learned components can significantly reduce collision checking and improve overall planning efficiency in challenging high-dimensional motion planning tasks.

2.1 Introduction

Sampling-based planning is a popular approach to high-dimensional continuous motion planning in robotics [96, 37, 79, 74, 51, 158, 95]. The idea is to iteratively sample configurations of the robots and construct one or multiple exploration trees to probe the free space, such that the start and goal states are eventually connected by some collision-free path through the sampled states, ideally with path cost minimized. This motion planning problem is hard, theoretically PSPACE-complete [138], and existing algorithms are challenged when planning motions of robots with a few degrees of freedom [89, 38, 11, 109, 28, 38]. In particular, the planning algorithms need to repeatedly check whether an edge connecting two sample states is feasible, i.e., that no state along the edge collides with any obstacle. This *collision checking* operation is the major computational bottleneck in the planning process and by itself NP-hard in general [75, 23]. For instance, consider the 7D Kuka arm planning problem in the environment shown in Figure 1. The leading sampling-based planning algorithm BIT* [51] spends about 28.6 seconds to find a complete motion plan for the robot, in which 20.2s (70.6% of time) is spent on collision checking. In comparison, it only takes 0.06s (0.2% of time) for sampling all the probing states needed for constructing random graphs for completing the search.

Learning-based approaches have become popular for accelerating motion planning. Many recent approaches learn patterns of the configuration spaces to improve the sampling of the probing states, typically through reinforcement learning or imitation learning [77, 71, 188, 133, 25]. For instance, Ichter et al. [71] and motion planning networks [133] apply imitation learning on collected demonstrations to bias the sampling process. The NEXT algorithm [25] provides a state-of-the-art design for embedding high-dimensional continuous state spaces into low-dimensional representations, while balancing exploration and exploitation in the sampling process. It has demonstrated clear benefits of using learning-based components to reduce samples and accelerate planning. However, we believe two aspects in NEXT can be improved, if we shift the focus from

reducing *sample complexity* to reducing *collision checking*. First, instead of taking the grid-based encoding of the entire workspace as input, we can use the graphs formed by batches of samples from the free space to better capture the geometric patterns of an environment. Second, having access to the entire graph formed by samples allows us to better prioritize edge exploration and collision checking based on relatively global patterns, and avoid getting stuck in local regions. In short, with reasonably relaxed budget of samples taken uniformly from the space, we can better exploit global patterns to reduce the more expensive collision checking operations instead. Figure 2.1 shows a typical example of how the trade-off benefits overall planning, and more thorough comparisons are provided in the experimental results section.

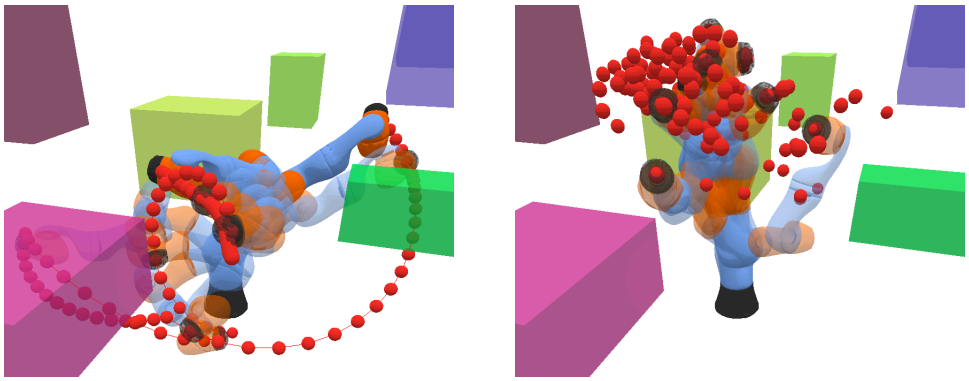


Figure 2.1: Performance on 7D Kuka arm. Left: Trajectory generated by the proposed GNN-based approach. Right: NEXT [25] getting stuck at a local region. Both methods were trained on the same training set.

We design two novel learning-based components that utilize Graph Neural Networks (GNNs) to accelerate the search for collision-free paths in batch sampling-based motion planning algorithms. The first component is the GNN Path Explorer, which is trained to find collision-free paths given the environment configuration and a random geometric graph (RGG) formed by probing samples. The second component is the GNN Path Smoother, which learns to optimize the path obtained from the explorer. In both models, we rely on the expressiveness and permutation invariance of GNNs as well as attention mechanisms to identify geometric patterns in the RGGs formed by samples, and accelerate combinatorial search.

The proposed learning-based components can accelerate batch sampling-based planning

algorithms without compromising probabilistic completeness properties. The methods achieve higher success rate, much lower rate of collision checking, and accelerate the overall planning compared to state-of-the-art methods. We evaluate the proposed approach in a variety of planning environments from 2D mazes to 14D dual KUKA arms. Experiments show that our method significantly reduces collision checking, improves the overall planning efficiency, and scales well to high-dimensional tasks.

2.2 Related Work

Learning-based Motion Planning. Learning-based approaches typically consider motion planning as a sequential decision-making problem, which could be naturally tackled with reinforcement learning or imitation learning. With model-based reinforcement learning, DDPG-MP [77] integrates the known dynamic of robots and trains a policy network. Strudel et al. [159] improves the obstacle encoding with the position and normal vectors. OracleNet [13] learns via oracle imitation and encodes the trajectory history by an LSTM [66]. Other than predicting nodes which are expected to greedily form a trajectory, various approaches have been designed to first learn to sample vertices, then utilize sampling-based motion planning for further path exploration through these samples. L2RRT [72] first embeds high-dimensional configuration into low-dimensional representation, then performs RRT [96] on top of that. Ichter et al. [71] uses conditional VAE to sample nodes. Zhang et al. [188] learns a rejection sampling distribution. Madaan et al. [112] encodes the explored tree with an RNN. Motion Planning Networks [133] utilizes the dropout-based stochastic auto-encoder for biased sampling. NEXT [25] projects the high-dimensional planning spaces into low-dimensional embedding discrete spaces, and further applies Gated Path Planning Networks [97] to predict the samples.

Existing learning-based approaches have considered improving *collision detection*. Fastron [33] and ClearanceNet [22] learn function approximators as a proxy to collision detection,

which is disparate from our focus on reducing the steps that are needed to the collision checker, and can be improved further potentially if combined together. Another recent line of work focuses on learning to explore edges given fixed graph. Value Iteration Networks [161] and Gated Path Planning Networks [97] apply convolutional neural networks (CNN) on discrete maps, then predict the policy with a weighted attention sum over neighborhoods. Generalized Value Iteration Networks [122] and Velickovic et al. [170] extend this approach for nontrivial graph by replacing CNN with GNN. However, the construction of such graphs requires ground-truth collision status for every edge on the graph at inference time.

It should be noted that other than sampling-based approaches, trajectory optimization [78, 194, 164, 149, 193], and motion primitives [118, 191] are standard choices for more structured problems such as for autonomous cars and UAVs, while sampling-based methods are important for navigating high-dimensional cluttered spaces such as for manipulators and rescue robots.

Graph Neural Networks for Motion Planning. Graph neural networks are permutation invariant to the orders of nodes on graph, which become a natural choice for learning patterns in graph problems. They have been successfully applied in robotics applications such as decentralized control [101]. For sampling-based motion planning, Khan et al. [83] utilizes GNN to identify critical samples. We focus on the different aspect of collision checking with given random geometric graphs, and can be combined with existing techniques without affecting probabilistic completeness. More broadly, GNNs have been used for learning patterns in general graph-structured problems, e.g. graph-based exploration [32, 150], combinatorial optimization [81, 40, 18], neural algorithm execution [170, 177, 180, 169]. Other than to use GNN for high-dimensional planning, several works propose to first learn neural metrics, then build explicit graphs upon the learned metric which is used later to search the path [146, 45, 43, 181]. While sharing similar interests, our work specifically focuses on how to reduce the collision checking steps for sampling-based motion planning.

Informed Sampling for Motion Planning. A main focus in motion planning is on developing

problem-independent heuristic functions for prioritizing the samples or edges to explore. Approaches include Randomized A* [37], Fast Marching Trees (FMT*) [74], Sampling-based A* (SBA*) [127], Batch Informed Trees (BIT*) [51]. These methods are orthogonal to our learning-based approach, which can further exploit the problem distribution and recognize patterns through offline training to improve efficiency. Recent work in motion planning has made significant progress in reducing collision checking through batch sampling and incremental search, such as in BIT* [51] and AIT* [158]. The idea is to start with batches of probing random samples in the free space, and focus on reducing collision checking to edges that are likely on good paths to the goal, which also inspires our work.

Lazy Motion Planning. Lazy motion planning also focuses on reducing collision checking, typically with hand-crafted heuristics or features. LazyPRM and LazySP [16, 58] construct an RGG first and check the edge lazily only if that edge is on the global shortest path to the goal. Instead of calculating a complete shortest path, LWA* uses one-step lookahead to prioritize certain edges [30]. LRA* interleaves between LazySP and LWA*, with a predefined horizon to lookahead [114]. These approaches use hand-crafted heuristics and do not utilize data-dependent information from specific tasks. Recently, GLS and StrOLL [113, 15] leverage experiences to learn to select the edge to check with either fixed graphs or hand-crafted features. Our GNN-based approach proposes the use of new representations for the learning-based components, with the goal of directly recognizing patterns using samples from the configuration space.

2.3 Preliminaries

Motion Planning. We focus on motion planning in continuous spaces, where the *configuration space* is $C \subseteq \mathbb{R}^n$. The configuration space includes all degrees of freedom of a robot (e.g. all joints of a robotic arm) and is different from the *workspace* where the robot physically resides in, which is at most 3-dimensional. For planning problem on a graph $G = \langle V, E \rangle$, we denote the start vertex

and goal vertex as $v_s, v_g \in C$. A path from v_s to v_g is a finite set of edges $\pi = \{e_i : (v_i, v_i')\}_{i\in[0,k]}$ such that $v_0 = v_s$, $v_k' = v_g$, and $v_i' = v_{i+1}$ for all $i \in [0, k-1]$. An environment for a motion planning problem consists of a set of obstacles $C_{obs} \subseteq C$ and free space $C_{free} = C \setminus C_{obs}$. Note that C_{obs} is the projection of 3D objects in the higher-dimensional configuration space, and typically has complex geometric structures that can not be efficiently represented. A sample state $v \in C$ in the configuration space is free if $v \in C_{free}$, i.e., it is not contained in any obstacle. An edge connecting two samples is free if $e \subseteq C_{free}$. Namely, for every point v on the edge e, $v \in C_{free}$. A path π is free if all its edges are free. A random geometric graph (RGG) is a r-disc or k-nearest-neighbor (k-NN) graph G [54, 179], where nodes are randomly sampled from the free space C_{free}. In this paper we consider the RGG as a k-NN graph. Given a random geometric graph G and a pair of start and goal configuration (v_s, v_g), the goal of agent is to find a free path π from v_s to v_g. Without loss of generality, we consider the cost of a path to be the total length over all edges in it.

Graph Neural Networks and Attention. Let $G = \langle V, E\rangle$ be a finite graph where each vertex v_i is labeled by data $x_i \in \mathbb{R}^n$. A graph neural network (GNN) learns the representation h_i of node v_i by aggregating the information from its neighbors $\mathcal{N}(v_i)$. Given fully-connected networks f and g, a typical GNN encodes the representation $h_i^{(k+1)}$ of the node v_i after k aggregation as:

$$c_i^{(k)} = \oplus^{(k)}(\left\{ f(h_i^{(k)}, h_j^{(k)}) \mid (v_i, v_j) \in E \right\}) \text{ and } h_i^{(k+1)} = g(h_i^{(k)}, c_i^{(k)}) \tag{2.1}$$

where $h_i^{(1)} = x_i$ and $\oplus$ is some permutation-invariant aggregation function on sets, such as max, mean, or sum. We will also use the attention mechanism when we need to encode a varied number of obstacles as inputs. Suppose there are n keys each with dimension d_q: $K \in \mathbb{R}^{n\times d_k}$, each key corresponding to a value $V \in \mathbb{R}^{n\times d_v}$. Given m query vectors $Q \in \mathbb{R}^{m\times d_k}$, we use a typical attention function $\text{Att}(K, Q, V)$ for each query as $\text{Att}(K, Q, V) = \text{softmax}(QK^T/\sqrt{d_k})V$ [168]. The function is also permutation-invariant so the order of obstacles does not affect the output.

2.4 GNN Architecture for Path Exploration and Smoothing

2.4.1 Overall Approach

At a high level, motion planning with batch sampling typically proceeds as follows [51]. We first sample a batch of configurations in the free space, together with the start and goal states, and form a random graph (RGG) by connecting neighbor states (such as k-NN). A tree is then built in this graph from the start towards the goal via heuristic search. The tree can only contain collision-free edges, so each connection requires collision checking. When a path from start to goal is found, it is stored as a feasible plan, which can be later updated to minimize cost. After adding all collision-free edges in the current batch in the tree, a new batch will be added and the tree is further expanded. The algorithm keeps sampling batches and expanding the tree until the computation budget is reached. It then returns the best path found so far, or failure if none has been found.

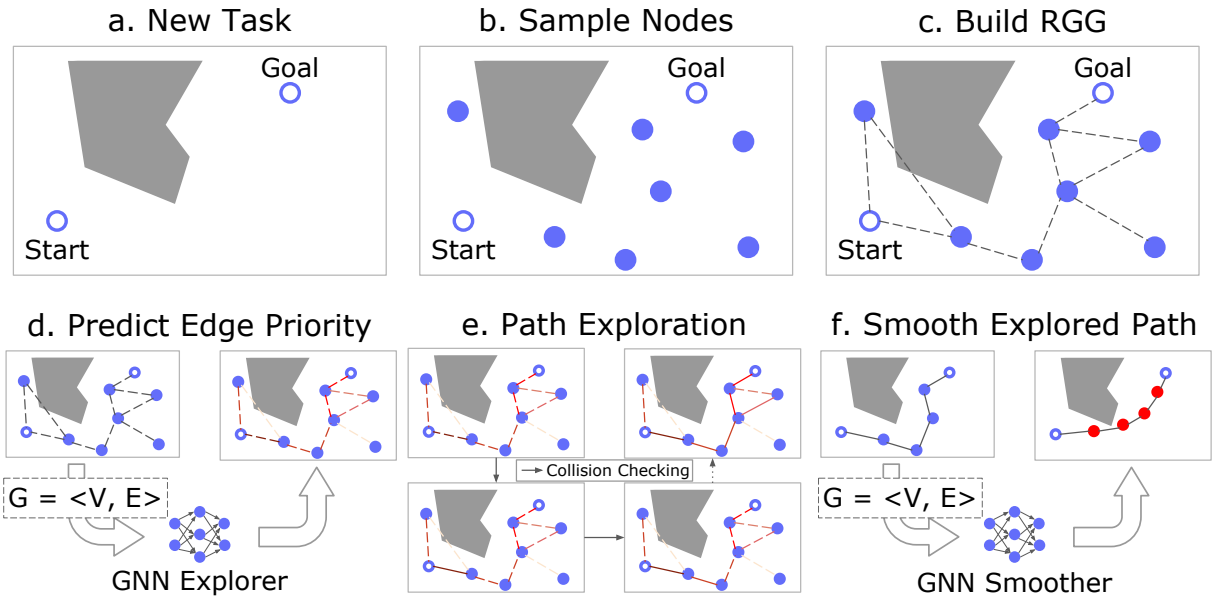


Figure 2.2: Main steps in planning with GNNs, as explained in Section 2.4.1.

We use GNN models to improve two important steps in this planning procedure: path exploration and path smoothing. The GNN path explorer finds a feasible path π from start to goal

given a randomly batch-sampled RGG, with the goal of reducing the number of edges that need to be checked for collision. The GNN path smoother then takes the path found by the explorer and attempts to produce another feasible path with less cost. In both tasks, the GNN models aim to recognize patterns and learn solutions to the combinatorial problems and save computation. In Figure 2.2, we illustrate the main steps for the overall algorithm. First, in (a-c), we generate an RGG composed of the vertices randomly sampled from the free space *without collision checking* on edges. This graph will provide patterns that the GNN explorer later uses to only prioritize certain edges for collision checking. In (d), the graph is the input to the GNN path explorer, which predicts the priority of the edges to explore and only the proposed edges are checked for collision. (e): We iteratively query the path explorer with collision checking to expand a tree in the free space until the goal vertex is reached, and sample new batches when no path is found in the current graph. (f): Once a path is provided by the path explorer, it becomes the input (together with the current RGG) to the GNN path smoother component, which outputs a new path that reduces path cost via local changes to the vertices in the input path.

2.4.2 GNN Architecture

We write the GNN path explorer as $\mathcal{N}_E$ and the GNN path smoother as $\mathcal{N}_S$. Both models take in a sampled random geometric graph $G = \langle V, E \rangle$. For an n-dimensional configuration space, each vertex $v_i \in \mathbb{R}^{n+3}$ contains an n-dimensional configuration component and a 3-dimensional one-hot label. There are 3 kinds of labels for the explorer: (i) the vertices in the free space, (ii) the vertices with collision, and (iii) the special goal vertex. There are also 3 kinds of labels for the smoother: (i) the vertices on the path, (ii) the vertices in the free space, and (iii) the vertices with collision.

The vertices and the edges are first embedded into a latent space with $x \in \mathbb{R}^{|V| \times d_h}, y \in \mathbb{R}^{|E| \times d_h}$, where d_h is the size of the embedding. The embeddings for the GNN explorer and smoother are different, which will be discussed later in this section. Taking the vertex and edge

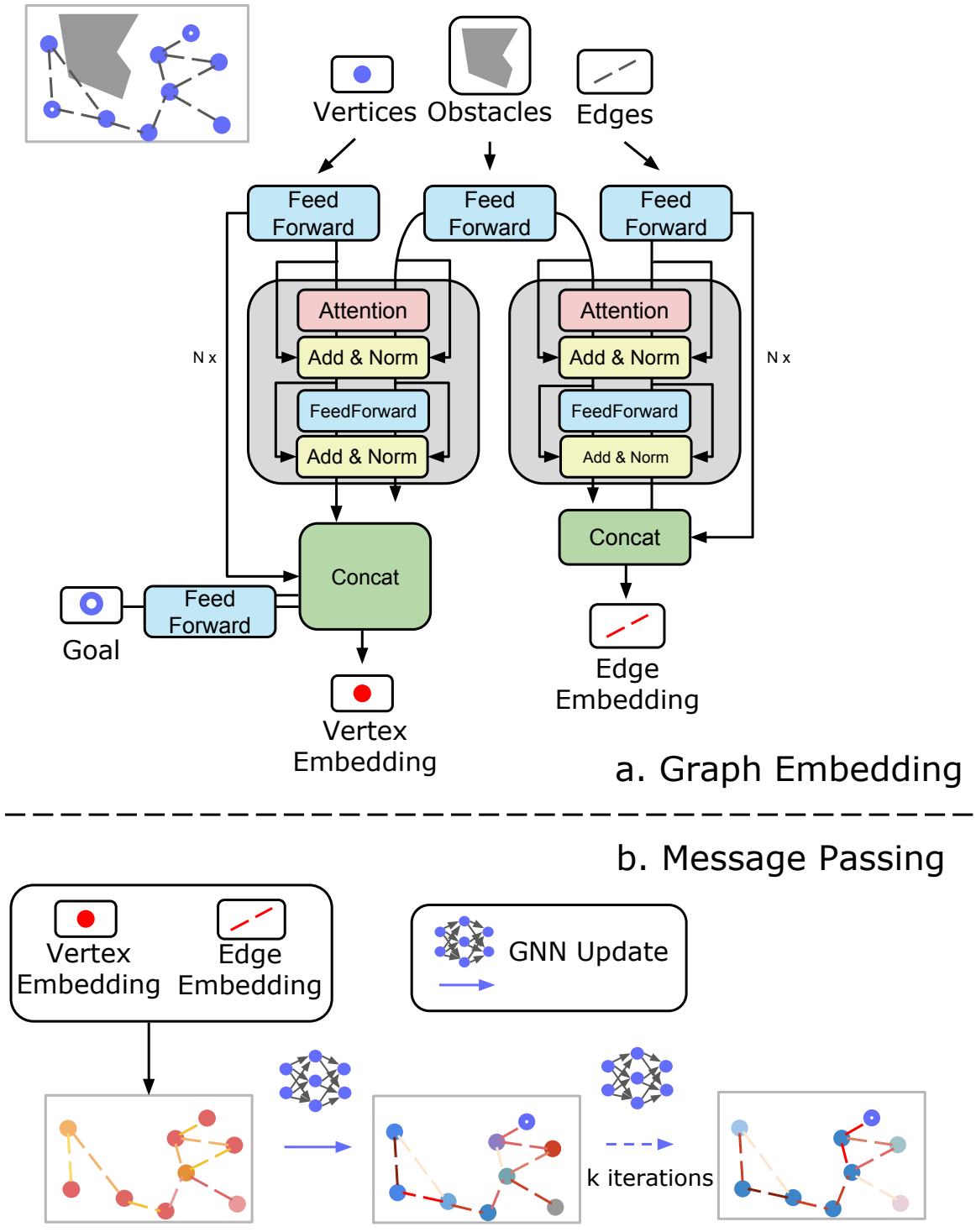


Figure 2.3: GNN architecture shared by the path explorer and the path smoother.

embedding x, y, the GNN aggregates the local information for each vertex from the neighbors, by performing the following operation with 2 two-layer MLPs f_x and f_y:

$$\begin{aligned} x_i &= g\left(x_i, \max\{f_x(x_j - x_i, x_j, x_i, y_l) \mid e_l : (v_i, v_j) \in E\}\right), \forall v_i \in V \\ y_l &= \max(y_l, f_y(x_j - x_i, x_j, x_i)), \forall e_l : (v_i, v_j) \in E \end{aligned} \tag{2.2}$$

Note that here we use max as the aggregation operator to gather the local geometric information, due to its empirical robustness to achieve the order invariance [130]. The edge information is also incorporated by adding y_l as the input to f_x. The update function g is implemented in two different ways for the GNN explorer and smoother. Specifically, g equals to the max operator for the GNN explorer, and $g(m_i, x_i) = f_g(m_i) + x_i$ as the residual connection for the GNN smoother, where f_g is a two-layer MLP. We choose max operator for the explorer, due to its inductive bias to imitate the value iteration, as mentioned by Velickovic et al. [170]. The residual connection is applied to the smoother, since intuitively the residual provides a direction for the improvement of each node on the path in the latent space, which fits our purpose to generate a shorter path for the smoother.

We also note that Equation 4.1 directly updates on the x and y and is a homogeneous function similar to Tang et. al [162], which allows us to self-iterate x and y over multiple loops without introducing redundant layers. Both the GNN explorer and smoother leverage this property. After several iterations, with two MLPs f_η, f_u, $\mathcal{N}_E$ outputs the priority $\eta = f_\eta(y)$ for each edge, and $\mathcal{N}_S$ outputs a potentially shorter path $\pi' = \{u_i, u_i'\}, u_i = f_u(x_i)$ for $v_i \in \pi$.

Obstacle Encoding

In the experiment part, we find obstacle encoding is helpful to the GNN explorer, which can optimize the explored path further with the smoother. We elaborate on the formulation of the obstacle encoding.

In this work, we consider an obstacle as a 2D or 3D box depending on the workspace,

denoted as $o = (p_1, \cdots, p_n, l_1, \cdots, l_n) \in \mathbb{R}^{2n}, n \in [2,3]$, where p_i and l_i are the center and length of the box along the i-th dimension. The environment configuration is written as $O \in \mathbb{R}^{|\{o\}| \times 2n}$, where $|\{o\}|$ is the number of the obstacles. Note that $|\{o\}|$ is a variable number, since the number of obstacles can be different for each problem.

Given MLPs $f_{a_x}^{(i)}, f_{a_y}^{(i)}, f_{K_x}^{(i)}, f_{Q_x}^{(i)}, f_{V_x}^{(i)}, f_{K_y}^{(i)}, f_{Q_y}^{(i)}, f_{V_y}^{(i)}$, the obstacle encoding is formulated as:

$$\begin{aligned} a_x &= \mathrm{LN}(x + Att(f_{K_x^{(i)}}(O), f_{Q_x^{(i)}}(x), f_{V_x^{(i)}}(O))) \text{ and } x = \mathrm{LN}(a_x + f_{a_x}^{(i)}(a_x)) \\ a_y &= \mathrm{LN}(y + Att(f_{K_y^{(i)}}(O), f_{Q_y^{(i)}}(y), f_{V_y^{(i)}}(O))) \text{ and } y = \mathrm{LN}(a_y + f_{a_y}^{(i)}(a_y)) \end{aligned} \tag{2.3}$$

where LN denotes the layer normalization [4]. This architecture follows the standard transformer block design [168].

Special design for the GNN path explorer.

The path explorer uses the embedding of the vertices of the form $x = h_x(v, v_g, (v - v_g)^2, v - v_g)$, where h_x is a two-layer MLP with batch normalization [73]. Here we append the L2 distance and the difference to the goal to the vertex embedding, which serve as heuristics for the GNN to be more informed about which node is more valuable. The y_l is simply computed as $y_l = h_y(v_j - v_i, v_j, v_i)$, where h_y is also a two-layer MLP with batch normalization. Optionally, it is helpful for the explorer to incorporate the configuration of obstacles $O = \{\mathbf{o}\} \in \mathbb{R}^{|\{\mathbf{o}\}| \times 2n}$ as inputs, when embedding the vertices and edges. Since the obstacles of the environment has variable numbers, we utilize the attention mechanism here to update the x and y, named as *obstacle encoding*, as illustrated in Figure 2.3.

As mentioned in the main part, the GNN explorer will update x and y over multiple loops. During training, we iterate x and y over a random number of loops between 1 and 10. Intuitively, taking random loops encourages the GNN to converge faster, which also helps propagating the gradient. During evaluation, the GNN explorer will output x and y after 10 loops. For loops larger

than 10, significant improvement on performance is not perceived. Finally, with an MLP f_η, the GNN explorer will output $\eta = f_\eta(y)$, which will be used as the priority to explore corresponding edges.

Special design for GNN path smoother.

The GNN smoother embeds vertices with $x = h_x(v)$, where h_x is a two-layer MLP with batch normalization. The y_l is computed as $y_l = h_y(v_j - v_i, v_j, v_i)$, where h_y is a two-layer MLP with batch normalization. Each time x and y are updated by Equation 4.1, the GNN smoother will output a new smoother path $\pi' = \{(u_i, u_i')\}_{i\in[0,k]}$, where $u_i = f_u(x_i), \forall v_i \in \pi$, given an MLP f_u. The u_0 and u_k' are manually replaced by v_s and v_g, to satisfy the path constraint. We assume the π' has the same number of nodes as π. Since the GNN smoother could gain novel local geometric information with the changed vertices of the new path, we dynamically update $G = \langle V, E\rangle$, via (i) replacing those nodes labeled as path nodes in V by the nodes on new path, (ii) replacing E by generating a k-NN graph on the updated V. With the updated graph G, we repeat the above operation. During training, the GNN smoother outputs π', after a random number of iterations (between 1 and 10). During evaluation, the GNN smoother outputs π' after only one loop for each calling.

2.5 Training the Path Explorer and Smoother

2.5.1 GNN Explorer $\mathcal{N}_E$: Training and Inference

The path explorer constructs a tree through sampled states with the hope of reaching the goal state in a finite number of steps. We initialize the tree $\mathcal{T}_0$ with the start state v_s as its root. Every edge $e_{\mathcal{T}_i}$ in the tree $\mathcal{T}_i$ exists only if $e_{\mathcal{T}_i}$ is in the free space C_{free}. Given an RGG $G = \langle V, E\rangle$, Our goal is to find a tree containing the goal configuration v_g by adding edges from E to the tree, with as few collision checks as possible. We write the edge on frontier of the tree as

$E_f(\mathcal{T}) = \{(v_i, v_i') \mid v_i \in V_\mathcal{T}, v_i' \notin V_\mathcal{T}\}$. We denote the set of edges with unknown collision status at time step i as E_i.

Training procedures. Each training problem consists of a set of obstacles O, start vertex v_s, goal vertex v_g, we sample a k-NN graph $G = \langle V, E \rangle$, where V is the random vertices sampled from the free space combined with $\{v_s, v_g\}$. The goal is to train $\mathcal{N}_E$ to predict exploration priority $\eta \in \mathbb{R}^{|E|}$.

A straightforward way for supervision is to use the Dijkstra's algorithm to compute the shortest feasible path from v_s to v_g, and maximize the corresponding values of η at the edges of this path, via cross entropy loss or Bayesian ranking [139]. However, it does not provide useful guidance when the search tree deviates from the ideal optimal path at inference time. Instead, we first explore the graph using η with i steps, which forms a tree $\mathcal{T}_i$, where i is a random number. The oracle provides the shortest feasible path π_N in this tree and connects one of the nodes on $\mathcal{T}_i$ to the goal vertex v_g. We formulate this optimal path as $\pi_N = \{e_{N_i} : (v_{N_i}, v'_{N_i})\}_{i \in [0,k]}$, where $v_{N_0} \in V_{\mathcal{T}_i}, v'_{N_k} = v_g$. We train the explorer to imitate this oracle. Namely, the explorer will directly choose $e_{N_0} \in \pi_N$ as the next edge to explore, among all possible edges on the frontier of $\mathcal{T}_i$, i.e. $E_i \cap E_f(\mathcal{T}_i)$. We maximize the η_{N_0} among the values of $\{\eta_i \mid e_i \in E_i \cap E_f(\mathcal{T}_i)\}$ using the following cross entropy loss:

$$L_{\mathcal{N}_E} = -\log \gamma_{N_0}, \text{ where } \gamma_k = \frac{e^{\eta_k}}{\sum_{e_j \in E_i \cap E_f(\mathcal{T}_i)} e^{\eta_j}}, \forall e_k \in E_i \cap E_f(\mathcal{T}_i) \tag{2.4}$$

Inference procedures. Given the GNN $\mathcal{N}_E$, the current explored tree $\mathcal{T}_i$ at step i, the RGG $G = \langle V, E \rangle$ including v_s and v_g, environment configuration O, GNN path explorer aims to maximize the probability of generating a feasible path by adding e_i from E_i to tree $\mathcal{T}_i$ as:

$$e_i = \underset{e_k \in E_i \cap E_f(\mathcal{T}_i)}{\arg\max} \; \mathcal{N}_E(\eta_k \mid V, E, O) \tag{2.5}$$

where η_k is the output of $\mathcal{N}_E$ for the edge e_k. After e_i is proposed by GNN using Equation 2.5, we check the collision of e_i. If e_i is not in collision with the obstacles, we add the edge e_i to the tree $\mathcal{T}_i$, and remove e_i from E_i, i.e., $E_{\mathcal{T}_{i+1}} = E_{\mathcal{T}_i} \cup \{e_i\}$, and $E_{i+1} = E_i \setminus \{e_i\}$. If e_i is in collision with obstacles, we query the path explorer for the next proposed edge using Equation 2.5, where E_i is updated as $E_i = E_i \setminus \{e_i\}$. The loop terminates when we find a collision-free edge, or when $E_i \cap E_f(\mathcal{T}_i) = \emptyset$. When the latter happens, we re-sample another batch of samples, add new samples to vertices V, re-construct k-NN graph G, re-compute η, and continue to explore the path on this new graph with the explored nodes and edges.

The exploration GNN only proposes an ordering on the candidate edges, and all possible edges may still be collision checked in the worst case. Thus, if there exists any complete path in the RGG, the algorithm always finds it. Therefore, the proposed learning-based component does not affect the probabilistic completeness of sampling-based planning algorithms [29].

2.5.2 GNN Path Smoother $\mathcal{N}_S$: Training and Inference

The GNN $\mathcal{N}_S$ for path smoothing takes an RGG and a path π proposed by the explorer, and aims to produce a shorter path π'. Specifically, the input is a graph $G = \langle V, E \rangle$, where $V = V_\pi \cup V_f \cup V_c$, $E = E_\pi \cup E_{fc}$. Here, V_f and V_c are reused as the same vertices in the GNN explorer, without introducing extra sampling complexity. E_π is composed of those pairs of the adjacent vertices on π, and E_{fc} connects each vertex in V_π to their k-nearest neighbor in $V_f \cup V_c$. Intuitively, aggregating information from $V_f \cup V_c$ can allow GNN to identify local regions that provide promising improvement on the current path, and avoid those that may yield potential collision.

Training procedures. We train the GNN path smoother $\mathcal{N}_S$ by imitating a smoothing oracle $\mathcal{S}$ similar to the approach of gradient-informed path smoothing proposed by Heiden et al. [62]. To prepare the training set, we iteratively perform the following two operations on each training

sample path. Given a feasible path π predicted by $\mathcal{N}_E$, the smoothing oracle first tries to move the nodes on the path π with perturbation within range ϵ. If the new path π_M is feasible and has cost less than π, then $\mathcal{S}$ will continue to smooth on π_M. Otherwise, $\mathcal{S}$ will continue smoothing on π via random perturbation. After several perturbation trials, the oracle further attempts to connect pairs of nonadjacent nodes directly by a line segment. If such a segment is free of collision, then the original intermediate nodes will be moved on this linear segment.

After $\mathcal{S}$ reaches maximum iteration, the oracle will return the smoothed path $\pi_M = \{w_i, w_i'\}_{i\in[0,k]}$. To generate training data, we first run our trained GNN explorer for each training problem to get the initial path π. Then the oracle path π_M and the predicted path π' are computed, and finally the GNN is trained via minimizing the MSE loss $L_{\mathcal{N}_\mathcal{S}} = \frac{1}{k}\sum_{i\in[1,k]} \|\mathcal{N}_\mathcal{S}(u_i \mid V,E) - w_i\|_2^2$.

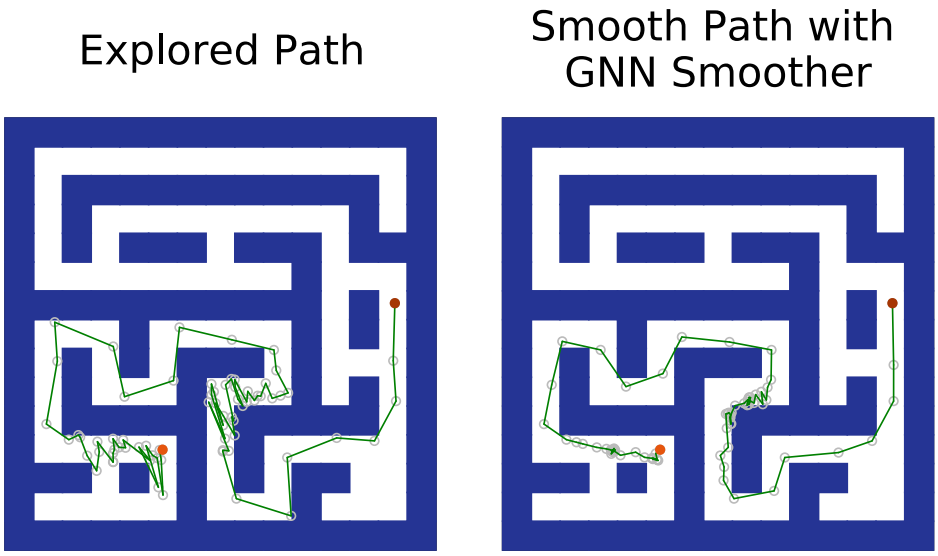


Figure 2.4: GNN path smoother on the 2D maze problem. It learns to improve the explored path and achieves lower cost.

Inference procedures. The GNN $\mathcal{N}_\mathcal{S}$ for path smoothing takes an explored path $\pi : \{(v_i, v_i')\}_{i\in[0,k]}$, a sampled graph G, and outputs a potentially shorter path $\pi' : \{(u_i, u_i')\}_{i\in[0,k]}$ which has the same number of edges as π. It is not always guaranteed that π' is collision-free. However, such π' still indicates directions for shortening the path, so we can improve π towards

π' in an incremental way. The GNN smoother will try to move every node v_i towards the target position u_i, with a small step size ϵ. For each time that all the vertices are moved with a small step, we check whether each edge still holds collision-free. If not, then the vertices on the edge will undo the movement. Otherwise, the new configuration of the vertex will replace its old configuration on π. This operation will be iterated over several times, until the maximum iteration is reached, or no edges on π can be moved further. We can then repeat the process by feeding the updated π back to the GNN $\mathcal{N}_S$ for further improvement. The intuition here is that there might still be chances to improve upon the updated π, by aggregating new information from its changed neighborhoods. This has shown empirical advantage in our experiments.

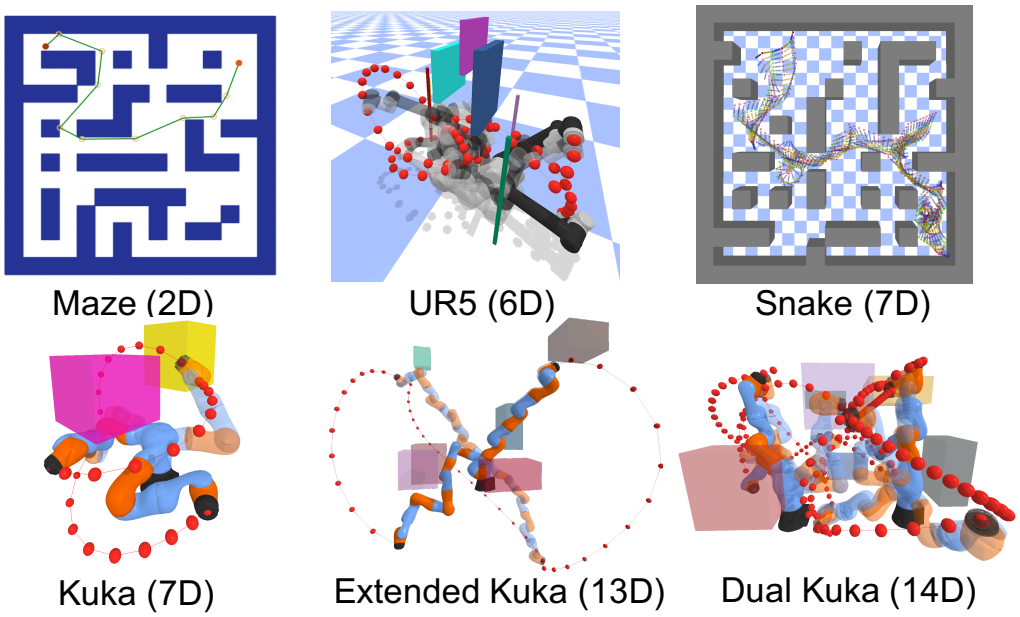


Figure 2.5: Demonstrations of all our environments.

2.6 Algorithms

Here we describe the full algorithms for the GNN path explorer and the GNN path smoother. The GNN path explorer is described in Algorithm 1. The GNN path smoother is described in Algorithm 2. The two algorithms are summarized in Figure 2.2.

The smoothing oracle that we use is similar to the approach of gradient-informed path smoothing proposed by [62]. Since the gradient in the configuration space is complex, we replace the gradient smoother by a random perturbation smoother. The oracle smoother jointly calls the random perturbation smoother and a segment smoother over multiple iterations. These two smoothers are described in Algorithm 3 and 4.

Algorithm 1 GNNExplorer

Input: obstacles O, start v_s, goal v_g, batch size n, node limit T_{max}
Sample n nodes from C_{free} to V_f
Sample n nodes from C_{obs} to V_c
Initialize $G = \{V : \{v_s, v_g\} \cup V_f \cup V_c, E : \text{k-NN}(V_f) \cup \text{k-NN}(V)\}$
Initialize $i = 0, E_0 = \text{k-NN}(V_f), V_{\mathcal{T}_0} = \{v_s\}, E_{\mathcal{T}_0} = \emptyset$
$\eta = \mathcal{N}_E(V, E, O)$
repeat
  select e_i with η
  $E_i \leftarrow E_i \setminus \{e_i\}$
  if $e_i : (v_i, v_i') \subseteq C_{free}$ **then**
    $V_{\mathcal{T}_{i+1}} \leftarrow V_{\mathcal{T}_i} \cup \{v_i'\}$
    $E_{\mathcal{T}_{i+1}} \leftarrow E_{\mathcal{T}_i} \cup \{e_i\}$
    $E_{i+1} \leftarrow E_i$
    $E_f(\mathcal{T}_{i+1}) \leftarrow \{e_j : (v_j, v_j') \in E_{i+1} \mid v_j \in V_{\mathcal{T}_{i+1}}, v_j' \notin V_{\mathcal{T}_{i+1}}\}$
    $i \leftarrow i + 1$
    if $||v_i' - v_g||_2^2 \leq \delta$ **then**
      $\pi \leftarrow$ path from v_s to v_g on tree $\mathcal{T}_i$
      return π
  if $E_i \cap E_f(\mathcal{T}_i) == \emptyset$ **then**
    Sample n nodes from C_{free}, add to V_f
    Sample n nodes from C_{obs}, add to V_c
    $V \leftarrow \{v_s, v_g\} \cup V_f \cup V_c$
    $E_i \leftarrow \text{k-NN}(V_f) \setminus (E \setminus E_i)$
    $E \leftarrow \text{k-NN}(V_f) \cup \text{k-NN}(V)$
    $\eta = \mathcal{N}_E(V, E, O)$
until $|V_f| > T_{max}$
return $\emptyset$

Algorithm 2 GNNSmoother

Input: step size ε, stop difference δ, outer loop L, inner loop K
Input: explored path $\pi : (v_i, v_i')_{i\in[0,k]}$, free samples V_f, collided samples V_c
for $n \in \{1 \dots L\}$ **do**
  $G \leftarrow \{V : \{V_\pi, V_f, V_c\}, E : \text{k-NN}(V_\pi, V) \cup E_\pi\}$
  $\pi' : (u_i, u_i')_{i\in[0,k]} = \mathcal{N}_S(V, E)$
  for $m \in \{1 \dots K\}$ **do**
    $d \leftarrow 0$
    for $u_i \in V_{\pi'}, i \in [1,k]$ **do**
      $w_i \leftarrow$ steer v_i toward u_i within step ε
      if $e_{i-1} : (v_{i-1}, w_i) \subseteq C_{free}$ **then**
        Replace $v_i \in \pi$ with w_i
        $d \leftarrow d + ||w_i - u_i||_2^2$
    if $d \leq \delta$ **then**
      break
return π

Algorithm 3 RandomSmoother

Input: path $\pi : (v_i, v_i')_{i\in[0,k]}$, perturbation range ε, iteration L_R
for $n \in \{1 \dots L_R\}$ **do**
  pick a random node $v_i \in \pi, 1 \leq i \leq k$
  $u_i \leftarrow v_i + \text{random}(-\varepsilon, \varepsilon)$
  if $(v_{i-1}, u_i), (u_i, v_{i+1}) \subseteq C_{free}$ and $Cost[(v_{i-1}, u_i), (u_i, v_{i+1})] < Cost[(v_{i-1}, v_i), (v_i, v_{i+1})]$ **then**
    replace v_i with u_i
return π

Algorithm 4 SegmentSmoother

Input: perturbed path $\pi : (v_i, v_i')_{i\in[0,k]}$ from Algorithm 3
$critical = [v_0, v_k']$
for $i \in [1,k]$ **do**
  if $(v_{i-1}, v_i') \not\subseteq C_{free}$ **then**
    append v_i to $critical$
$\pi_M = \emptyset$
for adjacent pair $v_i, v_j \in critical$ **do**
  $V \leftarrow \{v_p \mid v_p \in \pi, i \leq p \leq j\}$
  $E \leftarrow \{(v_a, v_b) \mid v_a, v_b \in V, (v_a, v_b) \subseteq C_{free}\}$
  $\pi_{ij} \leftarrow$ the shortest path from v_i to v_j via Dijkstra(V, E)
  $\pi_M \leftarrow \pi_M \cup \pi_{ij}$
return π_M

2.7 Experiments

2.7.1 Overall Performance

We compare our methods with the sampling-based planning baseline RRT* [79], the state-of-the-art batch-sampling based method BIT* [51], the lazy motion planning method LazySP [58], and the state-of-the-art learning-based method NEXT [25]. NEXT has been shown in [25] to outperform competing learning-based approaches. We conduct the experiment on 6 different environments, which are described in details as follows:

Maze The maze contains a 2D point robot. The datasets for training set and the test set for "Easy2D" is at `https://github.com/NeurEXT/NEXT-learning-to-plan/tree/master/algorithm` [25]. To generate the "Hard2D", we utilize the script provided by `https://github.com/RLAgent/gated-path-planning-networks` [97]. The Hard mazes are generated by controlling the obstacle density not less than 46%, and the distance from start to goal not less than 1.

UR5 The UR5 contains a UR5 robot arm [187], which has 6 degrees of freedom. There are two sets of boxes, poles and pads, which are set to generate in two different size range. The poles and pads are randomly generated in the workspace for each problem.

Snake The Snake environment contains a snake robot with 5 sticks, with another 2 degrees for the end position, which means 7D in total. The mazes are the same set of 2D mazes from NEXT.

Kuka, Extended Kuka The Kuka environment contains a 7DoF Kuka arm with fixed base position. The extended Kuka environment contains an extended 13DoF kuka arm. The boxes are randomly generated in the workspace for each problem.

Dual Kuka The environment contains two 7DoF KUKA arms, with 14DoF in total. Each arm need to reach the goal configuration, while required to not only avoid collision with the obstacles but also the other arm.

All the environments except the mazes are all implemented by PyBullet [31] with the MIT license.

For each environment, we randomly generate 2000 problems for training and 1000 problems for testing. Each problem contains a different set of random obstacles, and a pair of feasible v_s and v_g. We run all experiments over 4 random seeds. The averaged results are illustrated in Figure 2.6. For the 2D environment, we directly take the training set provided by NEXT to train our GNN. We use two test sets for the 2D environment: "Easy2D" is the original test set used for evaluating NEXT in the original paper [25], and "Hard2D" is a new set of tests we generated by requiring the distance between the start and goal to be longer than the easy environments. The goal is to test whether the learned models can generalize to harder problems without changing the training set.

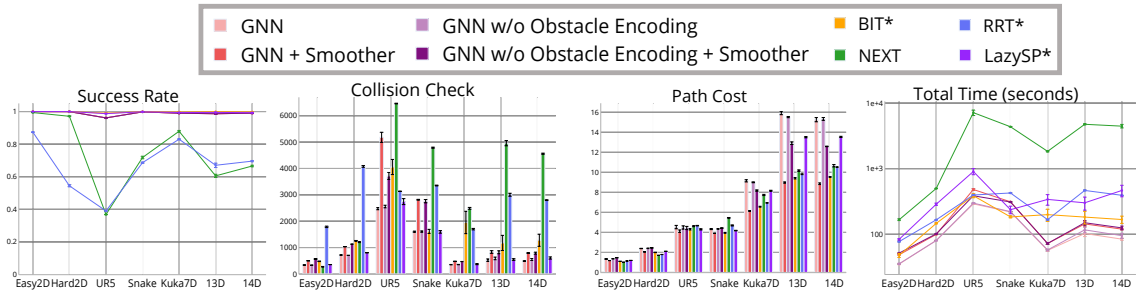


Figure 2.6: Comparison of performances on all environments from 2D to 14D, averaged over 4 random seeds. From left to right: (a) Success rate. (b) Collision checks. (c) Path cost. (d) Total planning time.

Success rate. As shown in Figure 2.6 (a), our method finds complete paths at 100.0% problems on both 2D Easy and 2D Hard, and at 97.18%, 99.85%, 99.20%, 99.15%, and 99.15% problems from UR5 to 14D, which is comparable to handcrafted heuristics used in BIT* (100.0%, 100.0%, 99.25%, 99.85%, 99.65%, 99.92%, 99.82% on each environment). The learning-based planner NEXT performs well on easy 2D problems (99.37% success rate), but drops slightly to 97.10%

on harder 2D problems, and 36.80%, 71.80%, 87.9%, 60.52%, 66.57% on environments in higher dimensions.

Collision checking. In Figure 2.6 (b), we see significant reduction of collision checking using the proposed approach, in comparison to other approaches especially in high-dimensional problems. The average number of collision checks by the GNN explorer is 336.3, 715.7, 2474.0, 1602.2, 350.5, 521.7, 487.0 on the environments, whereas BIT* needs 112%, 175%, 164%, 101%, 557%, 226%, 263% times as many collision checks as our method requires on each environments. LazySP needs 105%, 114%, 111%, 100%, 105%, 105%, 124% times as many collision checks as GNN requires on each environments. NEXT requires 270.23 checks on Easy2D and increases to 1206.1 on Hard2D. On the 14D environment, our method uses 17.4% of the collision checks as what NEXT requires.

Path cost. We show the average path cost over all problems where all algorithms successfully found complete paths. With the smoother and obstacle encoding, our GNN approach provides the best results from UR5 to 14D, and generates comparable results for the 2D Easy and 2D Hard, where NEXT is 1.02, 1.71, and GNN is 1.18, 2.05. We find that although the GNN explorer does not yield shorter path with obstacle encoding, these explored paths can be improved further with the smoother. The reason may be that with additional obstacle encoding, the GNN explorer tends to explore edges with less probability of collision and provides more space for the smoother to improve the paths.

Planning time. A common concern about learning-based methods is that their running cost due to the frequent calling of a large neural network model at inference time (as seen for the NEXT curve). We see in Figure2.6(d) that the wall clock time of using the GNN models is comparable to the standard heuristic-based LazySP, BIT* and RRT*, when all algorithms can find paths. The main reason is that the reduction in collision checking significantly reduces the overall time. We believe the GNNs can be further optimized to achieve faster inference as well.

In summary, experimental results show that the GNN-based approaches significantly

reduce collision checking while maintaining high success rate, low path cost, and fast overall planning. The performance scales well from low-dimensional to high-dimensional problems.

2.7.2 Tables for Overall Performance

Here we list the overall performances of all the methods on all the environments, including the averaged value with the standard deviation.

Table 2.1: Success rate. Our algorithm benefits from the probabilistic complete property from the RGG, which samples uniformly from free space,

	Easy2D	Hard2D	UR5	Snake	Kuka7D	13D	14D
GNN	**1.00±0.00**	**1.00±0.00**	0.96±0.00	**1.00±0.00**	0.99±0.00	0.99±0.00	0.99±0.00
GNN + Smoother	**1.00±0.00**	**1.00±0.00**	0.96±0.00	**1.00±0.00**	0.99±0.00	0.99±0.00	0.99±0.00
GNN w/o OE	**1.00±0.00**	**1.00±0.00**	0.96±0.00	**1.00±0.00**	0.99±0.00	0.99±0.00	0.99±0.00
GNN w/o OE + Smoother	**1.00±0.00**	**1.00±0.00**	0.96±0.00	**1.00±0.00**	0.99±0.00	0.99±0.00	0.99±0.00
BIT*	**1.00±0.00**	**1.00±0.00**	**0.99±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**
NEXT	0.99±0.00	0.97±0.00	0.37±0.00	0.72±0.01	0.88±0.01	0.61±0.01	0.67±0.00
RRT*	0.87±0.00	0.54±0.01	0.39±0.00	0.69±0.00	0.83±0.00	0.67±0.01	0.70±0.00
LazySP	**1.00±0.00**	**1.00±0.00**	**0.99±0.00**	**1.00±0.00**	0.99±0.00	0.99±0.00	0.99±0.00

Table 2.2: Collision check. GNN performs the best in most high dimensional problems.

	Easy2D	Hard2D	UR5	Snake	Kuka7D	13D	14D
GNN	336.25±3.71	715.65±6.76	**2474.03±40.35**	1602.16±22.66	**350.52±8.29**	**521.70±44.58**	**486.95±12.99**
GNN + Smoother	496.79±4.68	1029.72±8.33	5182.02±191.40	2813.75±15.01	477.32±9.06	830.95±49.06	791.78±14.27
GNN w/o OE	332.30±4.00	**703.72±5.78**	2556.63±49.91	1605.73±24.00	353.89±7.47	588.65±51.04	547.16±37.61
GNN w/o OE + Smoother	565.38±6.37	1126.02±9.91	3715.40±132.41	2757.88±62.64	466.06±6.70	820.70±55.97	789.25±36.54
BIT*	478.88±10.95	1253.56±15.38	4055.73±286.93	1612.22±78.85	1951.81±424.82	1175.42±287.68	1276.95±230.88
NEXT	**270.23±13.92**	1206.09±18.62	6461.13±14.31	4788.84±20.60	2488.49±33.76	4958.80±99.51	4559.99±21.92
RRT*	1785.46±27.93	4080.07±32.69	3135.36±4.03	3352.45±15.68	1698.04±28.34	3004.45±55.36	2796.99±13.89
LazySP	351.80±2.47	801.21±6.74	2742.12±113.08	**1595.74±48.15**	369.36±19.42	546.64±29.40	604.64±38.84

Table 2.3: Path cost. With the GNN smoother, our path cost is the lowest from UR5 to 14D.

	Easy2D	Hard2D	UR5	Snake	Kuka7D	13D	14D
GNN	1.34±0.01	2.39±0.02	4.54±0.17	4.33±0.05	9.15±0.11	15.91±0.15	15.26±0.21
GNN + Smoother	1.18±0.01	2.05±0.01	**4.12±0.12**	**3.91±0.01**	**6.14±0.02**	**8.98±0.06**	**8.86±0.08**
GNN w/o OE	1.36±0.01	2.41±0.03	4.50±0.15	4.35±0.03	9.00±0.04	15.52±0.06	15.34±0.14
GNN w/o OE + Smoother	1.46±0.01	2.45±0.03	4.45±0.15	4.43±0.02	8.18±0.06	12.91±0.13	12.59±0.03
BIT*	1.11±0.00	2.00±0.02	4.33±0.09	3.95±0.02	6.57±0.04	9.41±0.07	9.54±0.04
NEXT	**1.02±0.00**	**1.71±0.01**	4.62±0.05	5.45±0.04	7.74±0.06	10.17±0.07	10.66±0.12
RRT*	1.14±0.01	1.79±0.01	4.66±0.03	4.69±0.05	6.95±0.02	9.81±0.04	10.52±0.03
LazySP	1.20±0.01	2.11±0.01	4.30±0.06	4.18±0.03	8.16±0.05	13.51±0.06	13.54±0.06

Table 2.4: Total running time. GNN requires low time cost due to its optimization on collision checks.

	Easy2D	Hard2D	UR5	Snake	Kuka7D	13D	14D
GNN	**35.07±0.78**	80.57±1.16	**290.41±3.20**	218.83±3.41	**56.77±1.89**	**102.67±10.48**	**84.25±3.16**
GNN + Smoother	48.71±0.79	99.40±1.18	481.31±13.31	312.30±2.64	72.32±1.93	143.71±11.00	119.68±2.95
GNN w/o OE	35.43±0.66	**80.19±1.04**	298.18±4.25	221.26±2.29	57.62±1.87	116.69±11.75	95.19±8.36
GNN w/o OE + Smoother	51.59±0.76	101.40±1.17	386.30±9.97	310.42±4.39	72.39±1.85	149.86±12.19	125.09±8.25
BIT*	47.69±3.43	146.55±2.88	387.44±27.34	**183.84±6.69**	199.19±42.31	183.54±40.22	167.81±21.43
NEXT	166.46±4.81	499.65±4.40	7150.17±690.44	4355.78±28.44	1837.28±40.04	4750.26±88.29	4450.67±278.09
RRT*	77.20±1.03	166.06±1.35	396.86±0.45	425.87±2.46	166.98±2.43	465.80±8.58	392.57±2.74
LazySP	83.65±1.97	287.30±11.93	905.22±97.21	236.79±31.76	339.97±61.63	301.63±67.92	465.30±93.14

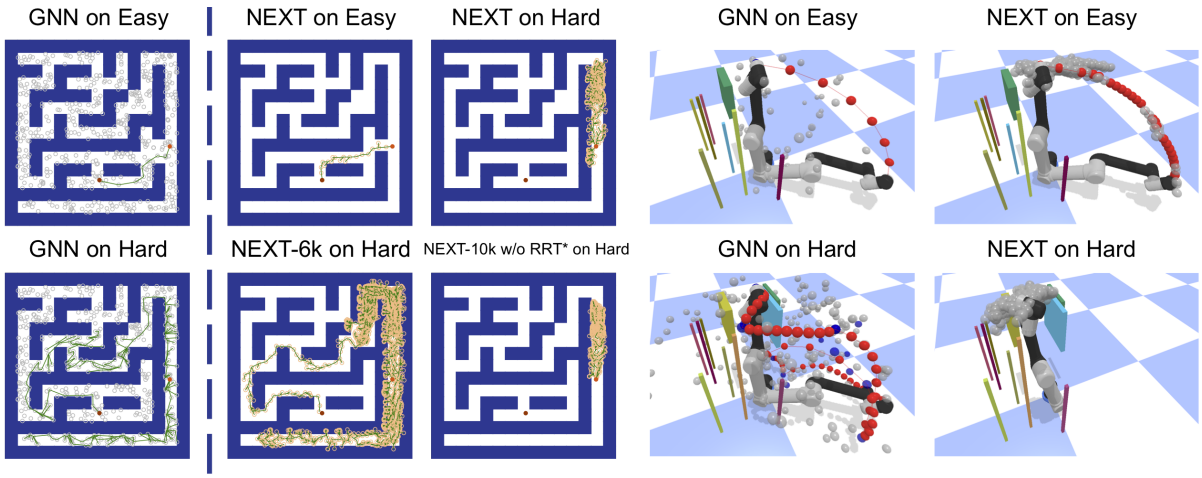


Figure 2.7: We test the generalization capability of GNN-based approaches and NEXT by constructing pairs of problems that have small but important difference in connectivity. The GNN models find paths on both environments quickly, while NEXT gets stuck in hard instances because of the lack of access to the graph structure provided by probing samples. The explored vertices of GNN on UR5 environment are colored in blue, and the edges on the path are colored in red.

2.7.3 Generalizability of Collision Reduction

A major challenge of learning-based approaches for planning is that small changes of the geometry of the environment can lead to abrupt change in the solutions, and thus lead the difficulty of generalization. In Figure 2.7, we provide evidence that the GNN approach can alleviate this problem because of its access to the graph structures formed by samples uniformly taken from the space. In the 2D maze environment, the top one is easy while the bottom one is much harder, although the difference is just whether a narrow corridor is present to the left of the start state. For the UR5 with pads and poles, to solve the hard one, the robot arm needs to first rotate itself around the z-axis to bypass the small pads, and then rotate it back to fit the

goal configuration. These problems are especially challenging for generalizing learned results to unseen environments.

We observe that the GNN-based components can handle the transition from the easy to hard problems consistently. In both environments, the GNN components find paths quickly, with 9 and 236 edges explored for 2D maze, and with 1 and 256 edges explored for UR5. In contrast, the NEXT model trained on the same training sets, can quickly find a path in the easy problem, but gets stuck in the hard one. Since the two problems are close to each other in the input space for NEXT, it is not surprising to see this difficulty of generalization. In fact, the only case where NEXT can eventually find a path on 2D Maze after more than 6K edge exploration is when the algorithm delegates 10% operations to standard RRT*. Without delegating to RRT*, NEXT gets stuck in local regions after 10K exploration steps on 2D and 1K steps on UR5.

2.7.4 Further Comparison with Lazy Motion Planning

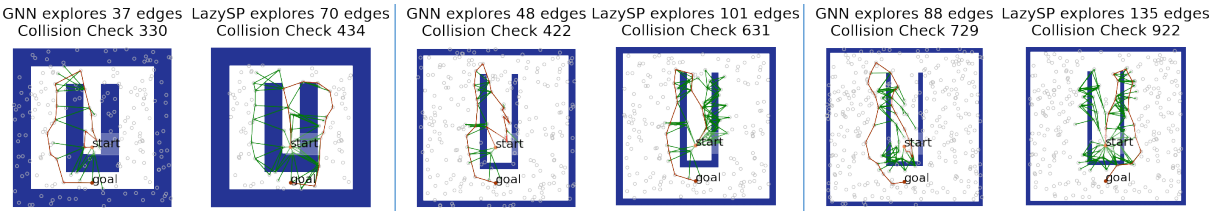


Figure 2.8: Comparison of performances with LazySP across different batch samples. The final path is colored red, and the other explored edges are colored green. From left to rightm we show examples of the search results with the batch size being 100, 200, and 300, respectively. We show obstacles of different thickness because the in-collision samples are also part of the inputs of the GNN planner, which can provide useful information about the topology of the environment.

We compare GNN with the lazy motion planning method LazySP [58]. Lazy approaches prioritize the collision checking on edges that are part of the shortest paths in the RGG, which is a strategy that can often see good performance especially on randomly generated graphs. However, as lazy planning uses the fixed heuristic of prioritizing certain paths, it is easy to come up with environments where this heuristic becomes misleading. Our proposed learning-based

approach, instead, uses GNN to discover patterns from the training set, and can thus be viewed as a data-dependent way of forming heuristics for reducing collision checking.

Consider U-shaped obstacles, with the start state close to the bottom of the U-shape, as shown in Figure 2.8. It is a standard environment that is particularly hard for the standard lazy approach, which prioritize the edges that directly connect the start and goal states, as they are close in the RGG that does not consider obstacles. Indeed, the lazy approach needs to check most of the edges that cross the obstacles before the path for getting out of the U-shape can be found. We train the GNN-based model on these environments and observe clear benefits of learning-based components in avoiding this issue. Figure 2.8 shows the difference between the two approaches in several examples of the 2D environment using different sizes of sample batches, where the GNN-based approach can typically save 50% of collision checking on edges compared to the lazy approach.

2.7.5 Ablation Study: Probing Samples

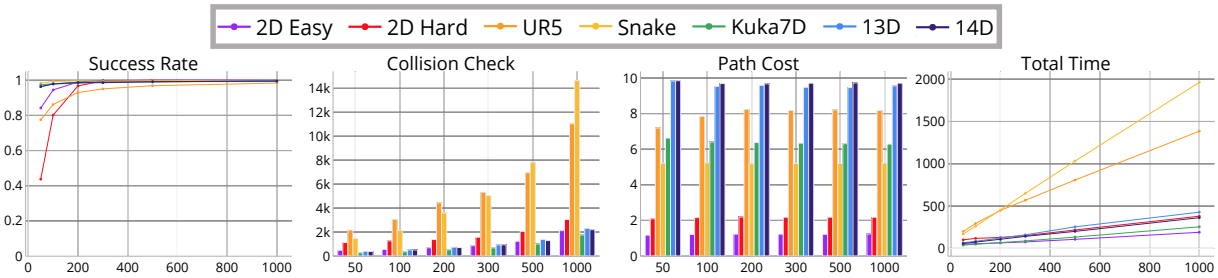


Figure 2.9: Comparison of performances for different probing samples from 2D to 14D problems.

In the following subsections, we perform ablation studies of varying different parameters in our approach. The first study is that we use RGGs with different number of vertices from the free space, using 100, 200, 300, 500, 1000 samples, as illustrated in Figure 2.9. The success rate increases when there are more probing samples, which is consistent with the resolution-complete property of sampling-based planning. The path cost stays nearly the same for all

settings, indicating robustness of the smoother models. The collision checks and planning time grows linearly with the probing samples. The reason is that the number of edges of k-NN RGG increments linearly with vertices, the input to the GNN grows linearly, thus the computation cost increases linearly on both CPU and GPU.

2.7.6 Ablation Study: Varying Training Set Size for Explorer

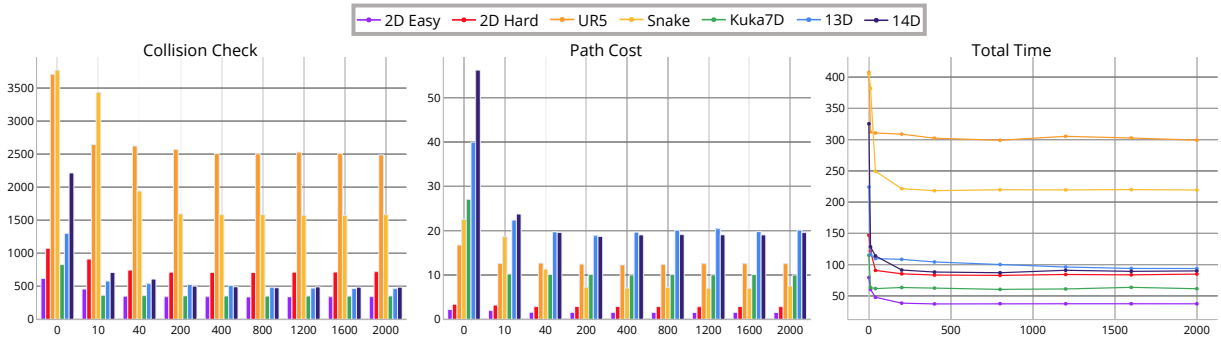


Figure 2.10: Ablation study on the different training set size. All the performances become stable at relatively few training set size (around 40 problems, 2% of the original training set).

We conduct further experiments to analyze the effect of the training set size. We train the GNN path explorer with the 0 (0%), 10 (0.5%), 40 (2%), 200 (10%), 400 (20%), 800 (40%), 1200 (60%), 1600 (80%), 2000 (100%) problems in this new training set. The performance of collision checks, path costs, and total time on the testing problems are demonstrated in Figure 2.10.

As shown, the overall performance of the GNN explorer is robust, even with 2% problems of the original training set. There are two reasons here: (i) The GNN models we propose are relatively lightweight in terms of the parameter numbers, which means that it is suffice to train it with small amount of data. (ii) Our GNN model does not depend on the global feature of the whole graph, as it only aggregates the information from local neighborhoods. Though each problem yields a different graph in terms of global characteristic, they can share similar local geometric patterns, which is beneficial for the efficiency of learning GNN models.

2.7.7 Ablation Study: Feature Choices

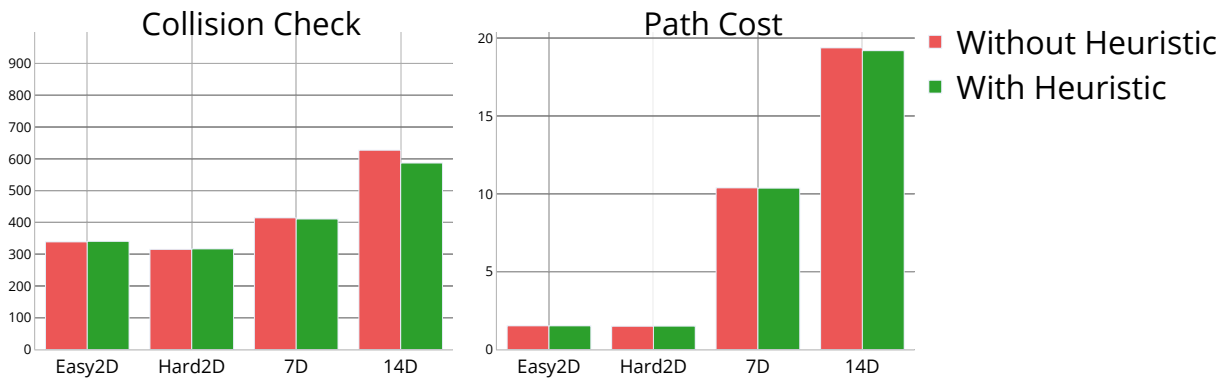


Figure 2.11: Comparison of performance of GNN explorers with vertex embedding x as $h_x(v, v_g)$ and $h_x(v, v_g, (v - v_g)^2, v - v_g)$ respectively. Results show that the GNN explorer with heuristics perform slightly better for high-dimensional problems.

In this experiment, we replace the vertex embedding $x = h_x(v, v_g, (v - v_g)^2, v - v_g)$ by $x = h_x(v, v_g)$, which removes the L2 distance heuristic on features. We retrain the new GNN with the same training set, and compare to the original architecture on 4 environments. As shown in Figure 2.11, the performances of two GNNs are close to each other from 2D to 7D environments (0.5%, 0.6%, 1.0% in terms of collision checking), and the original GNN is slightly better on 14D environment (6.8% in terms of collision checking, 0.9% in terms of path cost). The L2 distance heuristic is helpful in high dimensions, but does not have much effect, due to the complex geometry of the C-space.

2.7.8 Ablation Study: GNN Smoother Versus Oracle Smoother

In this experiment, we replace the learned GNN smoother by the oracle smoother, which is the expert that GNN smoother imitates. We compare these two smoothers, given the same path explored by the GNN explorer on the test problems. As shown in Figure 2.12, our smoother requires much fewer collision checks and time, while maintaining comparable improvement on the explored path, which is contributed by the incremental way of smoothing, and the generalizablity of the GNN. More specifically, the GNN smoother requires 3.9%, 8.8%, 10.5%, 8.0%, 1.9%,

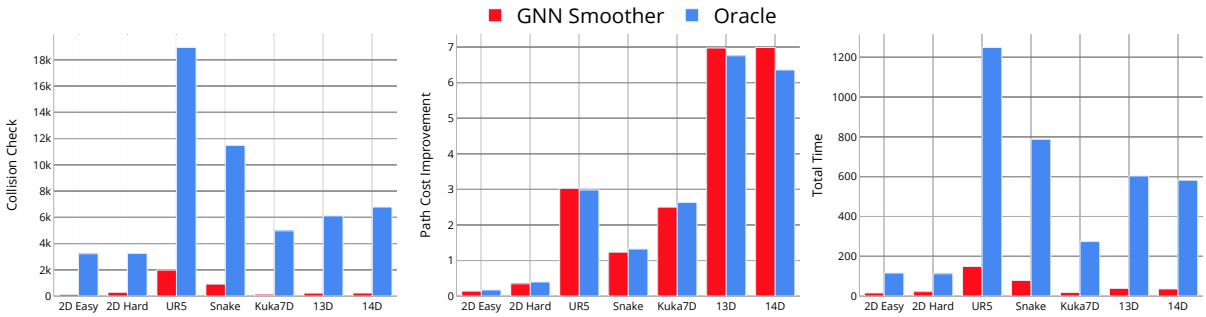


Figure 2.12: Comparison of performance between our GNN smoother and the oracle to trained on. Our GNN smoother learns to smooth the path with comparable improvement as the oracle, and also requires fewer collision checking steps and less total time.

3.7%, 3.5% as many collision checks as the oracle on each environment, while maintaining 80.7%, 86.9%, 101.5%, 93.2%, 94.9%, 103.2%, 109.9% as much improvement as the oracle for each environment.

2.7.9 Ablation Study: Varying the k in k-NN

As suggested in [179], we set the k in the k-NN graph as proportional to the logarithm of the number of vertices, which is formulated as $\lceil k_0 \cdot \frac{\log |V_f|}{\log 100} \rceil$. Here we test different k_0 for the k-NN, choosing among $\{1, 2, 4, 10, 20, 40\}$, as demonstrated in Figure 2.13.

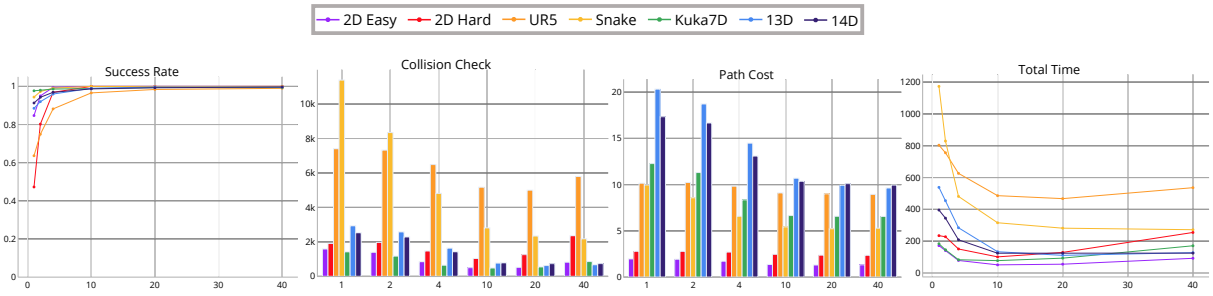


Figure 2.13: Ablation study on different k_0 for k-NN. Performances are best at $k_0 \in [10, 20]$.

It is not surprising for the success rate to increase when k increases, since there are more edges in the graph, which increases the possibility to find a feasible path. The path cost also decreases with increasing k, since on average there might be fewer segments on a path. The collision checks and total running time first decrease then increase, since larger k brings higher

possibility to find a feasible path with fewer intermediate vertices, but also brings more edges to check on the frontier.

2.7.10 Ablation Study: Varying the Batch Size

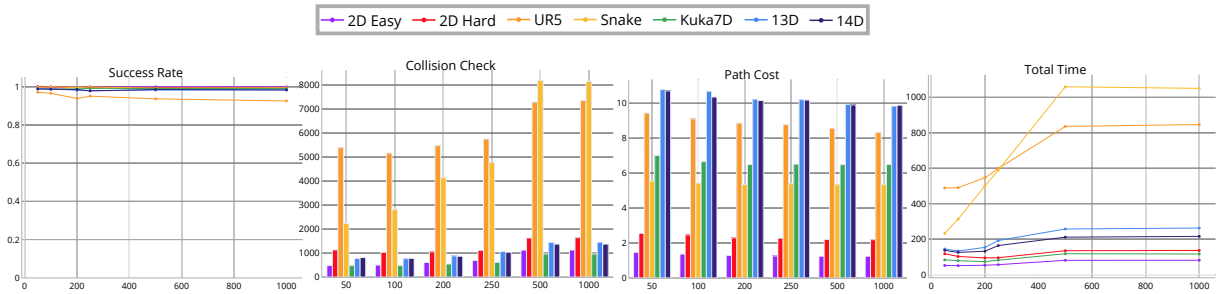


Figure 2.14: Ablation study on different batch size for batch sampling. Larger batches tend to yield lower success rate and path costs, while requiring more collision checks and running time.

In this experiment, we inspect the effect of the batch size. While constraining the maximum sampling number from free space to be 1000, we set the batch sampling size among $\{50, 100, 200, 250, 500, 1000\}$. We see that the success rate drops when the batch size increases, since the GNN explorer is given fewer opportunities to fail and re-sample. The collision checks grows with the batch size, because the graphs would contain more edges with larger batch size on average. The path cost is lower with larger batches, similar to the effect of higher k, due to higher possibility to find a feasible path with fewer intermediate vertices. The total time raises when the batch size goes larger, because larger batches brings denser graphs, which enlarges both CPU and GPU costs.

2.8 Conclusion

We presented a new learning-based approach to reducing collision checking in sampling-based motion planning. We train graph neural network (GNN) models to perform path exploration

and path smoothing given the random geometric graphs (RGGs) generated from batch sampling. We rely on the ability of GNN for capturing important geometric patterns in graphs. The learned components can significantly reduce collision checking and improve overall planning efficiency in complex and high-dimensional motion planning environments.

2.9 Acknowledgments

Chapter 2, in full, is a reprint of Chenning Yu, Sicun Gao, "Reducing Collision Checking in Sampling-based Motion Planning", NeurIPS, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Accelerating Multi-Agent Planning Using Graph Transformers with Bounded Suboptimality

Conflict-Based Search is one of the most popular methods for multi-agent path finding. Though it is complete and optimal, it does not scale well. Recent works have been proposed to accelerate it by introducing various heuristics. However, whether these heuristics can apply to non-grid-based problem settings while maintaining their effectiveness remains an open question. In this work, we find that the answer is prone to be no. To this end, we propose a learning-based component, i.e., the Graph Transformer, as a heuristic function to accelerate the planning. The proposed method is provably complete and bounded-suboptimal with any desired factor. We conduct extensive experiments on two environments with dense graphs. Results show that the proposed Graph Transformer can be trained in problem instances with relatively few agents and generalizes well to a larger number of agents, while achieving better performance than state-of-the-art methods.

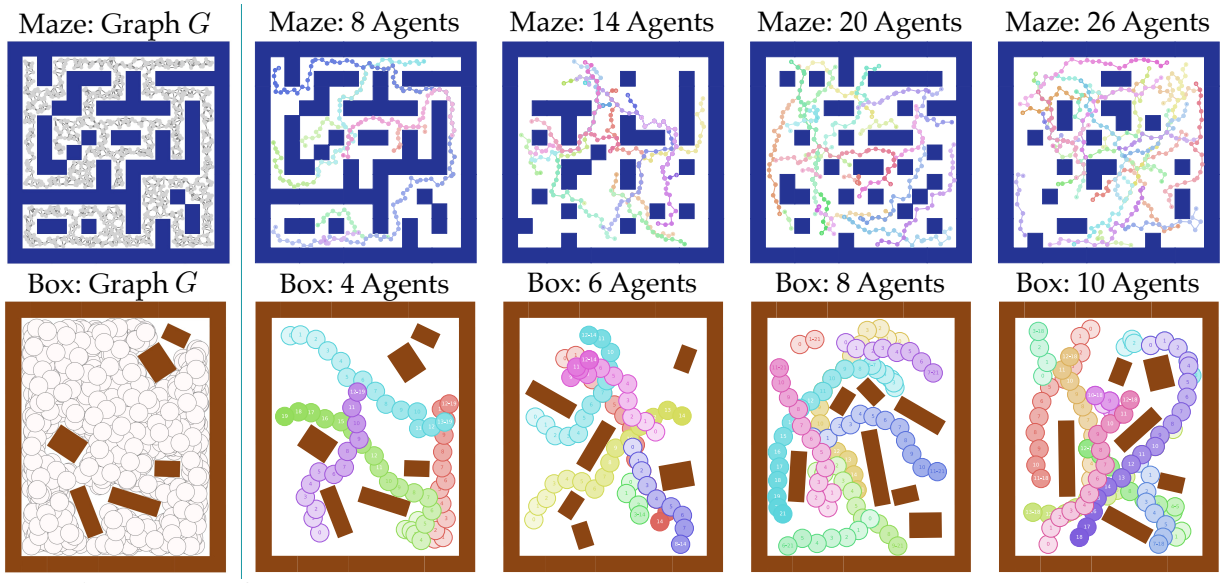


Figure 3.1: **Left:** Examples of our graph-based MAPF instances. To construct the graph, we sample vertices randomly from the free space and connect them with collision-free edges. **Right:** Problem instances that our approach solves while other baselines fail. Different colors represent the trajectories of different agents. Vertices in the same trajectory have deeper colors if their respective time steps are later.

3.1 Introduction

Multi-Agent Path Finding (MAPF) is central to many multi-agent problems. The solution to MAPF is to generate collision-free paths guiding agents from their start positions to designated goal positions. MAPF has practical applications in item retrieval in warehouses [44], mobility-on-demand services [129], surveillance [56] and search and rescue [92].

Conflict-Based Search (CBS) is one of the most popular planners for MAPF [151]. It is provably complete and optimal. However, solving MAPF optimally is NP-hard [6, 184]. Consequently, CBS suffers from scalability, as the search space grows exponentially with the number of agents. Bounded-suboptimal algorithms [9, 100] guarantee a solution that is no larger than a given constant factor over the optimal solution cost. Though these methods often run faster than CBS for grid-based MAPF instances, their effectiveness remains an open question for non-grid-based problem settings, wherein agents can move in an arbitrary continuous domain. In addition, since most of these heuristics rely heavily on collision checking for conflicts, their

computational costs may become considerable when the graphs are dense.

Recently, learning-based methods have shown their potential in solving MAPF tasks efficiently [101, 70, 69, 145], which offload the online computational burden into an offline learning procedure. Yet, we find nearly none of them address graph-based MAPF settings. Therefore, the main interest of this work is to explore whether and how a learning-based method could accelerate MAPF planners in non-grid-based settings, especially for dense graphs.

Contributions. We propose to use a Graph Transformer as a heuristic function to accelerate Conflict-Based Search (CBS) in a non-grid setting. Similar to previous works [69], by introducing focal search to CBS, our framework guarantees both the completeness and bounded-suboptimality of the solution. Our contributions are as follows:

- We propose a novel architecture, i.e., the Graph Transformer, which leverages the underlying structure of the MAPF problem. The proposed architecture has several desired properties, e.g., dealing with an arbitrary number of agents, making it a natural fit for the MAPF problem. To our knowledge, our work is one of the first works to introduce a learning component to MAPF problems under *non-grid-based* problem settings.

- We design a novel training objective, i.e., Contrastive Loss, to learn a heuristic that ranks the search nodes. Unlike [69], our loss can be directly optimized without introducing an upper bound, which is suitable for deep learning.

- We demonstrate the generalizability of our model by training with relatively few agents and testing in unseen instances with larger agent numbers. Results show that our approach can accelerate CBS significantly while ECBS, using handcrafted heuristics [9], fails.

Related Work. Leading MAPF planners mainly include three types: optimal planners, bounded-suboptimal planners, and unbounded-suboptimal planners. Optimal planners include BCP solvers [93, 94], and Conflict-Based Search (CBS) [151], followed by its variants, e.g., CBSH [47] and CBSH2 [99]. Bounded-suboptimal planners are another line of work with

better scalability while guaranteeing completeness and bounded-suboptimality. Representative works include EPEA* [46], A* with operator decomposition [156], M* [171], ECBS [9], and EECBS [100]. Last but not least, there are unbounded-suboptimal planners that aim to aggressively accelerate the planning. Examples include Prioritized Planning [165], ORCA [166], Push-and-Swap [108], and Parallel Push-and-Swap [143]. These works can find solutions fast, but do not guarantee the solution quality [21, 167].

Recently, learning-based methods were introduced to solve multi-agent tasks efficiently, using imitation learning [101, 102] and reinforcement learning [145, 173, 183]. These end-to-end methods are often good at memorizing the patterns that are seen during training, which could save significant online computation when deployed to similar tasks. However, it is hard to ensure their completeness. To this end, several works have been proposed to train a learning-based heuristic and use it to guide the tree search [82, 189, 154, 182]. To accelerate CBS and ECBS, Huang et al. [70, 69] use Supported Vector Machines (SVM) to bias the search, and train it via imitation learning. Our work is not only one of the first works that apply ML to *non-grid-based* problem settings, but also one of the first works that apply *deep learning* to MAPF with completeness and bounded-optimality guarantees.

3.2 Problem Formulation

We study Multi-Agent Path Finding (MAPF) in the 2D continuous space $\mathcal{C} \subseteq \mathbb{R}^2$. The configuration space $\mathcal{C}$ consists of a set of obstacles $\mathcal{C}_{obs} \subseteq \mathcal{C}$ and free space $\mathcal{C}_{free} : \mathcal{C} \setminus \mathcal{C}_{obs}$. Note that $\mathcal{C}_{obs}$ could be different from what appears in the workspace, since it also considers the geometric shape of the agent, which may not solely be a point mass.

A random geometric graph $G = \langle V, E \rangle$ is sampled from the space. Every sampled vertex $v \in V$ is collision-free, i.e., $v \in \mathcal{C}_{free}$. A directed edge $e \in E : (v_i \rightarrow v_j)$ connects v_i to v_j, if (i) v_j is one of the neighbors of v_i, and (ii) the edge is collision-free, i.e., $e \subseteq \mathcal{C}_{free}$. The neighbor set

can be defined as the r-radius or k-nearest neighbors.

Suppose there are M agents on this graph G. Each agent i occupies a region $\mathcal{R}(q) \subseteq \mathcal{C}$, associated with a vertex $q \in V$. We assign a start vertex s_i and a goal vertex g_i to each agent i. We denote the path of agent i as $\sigma_i : \{v_i^t\}_{t \in [1 \cdots T_i]}$, where $T_i \in \mathbb{Z}_{>0}$, and agent i is on vertex v_i^t at time step t. We denote e_i^t as the edge $(v_i^t \rightarrow v_i^{t+1})$ that traverses from v_i^t to v_i^{t+1} in 1 timestep.

Problem Description. We consider a tuple $(G, \mathcal{S}, \mathcal{G}, \mathcal{C}, \mathcal{R})$ as a problem instance of MAPF, where $\mathcal{S} : \{s_i\}_{i \in [1 \cdots M]}$ and $\mathcal{G} : \{g_i\}_{i \in [1 \cdots M]}$ are the start and goal vertices. A conflict-free solution $\{\sigma_i\}_{i \in [1 \cdots M]}$, should satisfy the following objectives, given arbitrary time t and pair of agents i, j [123]:

(Endpoint) $v_i^0 = s_i \wedge v_i^{T_i} = g_i$

(Obstacle) $v_i^t \in \mathcal{C}_{free} \wedge e_i^t \subseteq \mathcal{C}_{free}$

(Inter-agent) $\mathcal{R}(q_i) \cap \mathcal{R}(q_j) = \emptyset, \forall q_i \in e_i^t, q_j \in e_j^t$

Note: If $t \geq T_i$, we assume the corresponding v_i^t is equal to $v_i^{T_i}$. This means that the agent will stay at the goal starting from time step $v_i^{T_i}$.

Solution Quality. We assume each edge requires 1 time step to traverse. The quality of the solution is measured by the sum of travel times (flowtime): $\sum_{i \in [1 \cdots M]} T_i$.

3.3 Background: Conflict-Based Search with Biased Heuristics

In this section, we first introduce Conflict-Based Search (CBS), an optimal multi-agent planner [151]. Then we introduce *focal search* [126, 53], which incorporates the biased heuristic into the CBS framework, while preserving the guarantees of bounded-suboptimality and completeness [9].

3.3.1 Conflict-Based Search

Conflict-Based Search (CBS) is an optimal bi-level tree search algorithm of MAPF. The high-level planner aims to solve inter-agent conflicts, while the low-level planner aims to generate optimal individual paths. Here we denote an inter-agent conflict as $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, which implies two edges, $(v_i^{t-1} \rightarrow v_i^t)$ and $(v_j^{t-1} \rightarrow v_j^t)$, dissatisfy the inter-agent objective mentioned in Section 3.2.

The high-level planner maintains a tree and decides which search node to expand in a best-first manner. To this end, each search node N stores the following information:

(1) A set of constraints $\mathcal{T}(N)$. A constraint (i, v, t) indicates that agent i should not traverse to graph vertex v at time t.

(2) A solution $\sigma(N)$: $\{\sigma_i\}_{i\in[1\cdots M]}$. The solution satisfies the endpoint and obstacle objective, but may or may not satisfy the inter-agent objective. In addition, the solution should obey the constraints $\mathcal{T}(N)$, i.e., $\forall i, \forall v_i^t \in \sigma_i, (i, v, t) \notin \mathcal{T}(N)$.

(3) The cost of the solution $c(N)$. The high-level planner prioritizes which search node to expand based on this metric.

On the high level, CBS first creates a root search node with no constraints, then keeps selecting a search node and expanding it. A search node N^* is selected if it is a leaf node with the lowest cost. CBS then checks whether the solution $\sigma(N^*)$ has an inter-agent conflict. If there is no conflict, $\sigma(N^*)$ will be returned as the final result. Otherwise, CBS chooses the first conflict C: $(i, j, t, v_i^{t-1}, v_j^{t-1}, v_i^t, v_j^t)$, and splits it into two constraints C_1: (i, v_i^t, t) and C_2: (j, v_j^t, t). Two child search nodes N_1 and N_2 are then generated, with constraints $\forall i = 1, 2, \mathcal{T}(N_i) = \mathcal{T}(N^*) \cup \{C_i\}$ respectively. Then an optimal low-level planner, e.g., A* [60], is called by each child search node, which replans the path for each affected agent and records the respective solution and cost. CBS guarantees completeness and optimality, since both the high-level and low-level planners are performing best-first search [151].

3.3.2 Incorporating Biased Heuristics using Focal Search

CBS is an optimal planner, but it does not scale well even for grid-based problems settings. To improve the scalability, focal search [126, 53] was introduced by previous works, e.g., Bounded CBS (BCBS) and Enhanced CBS (ECBS) [9]. Here we describe a simplified version of BCBS.

We present the pseudocode of focal search in Algorithm 5. Focal search introduces the focal set to the CBS framework. A focal set (*Focal*) maintains a fraction of the leaf search nodes in the CBS tree (i.e., *Open*). We denote *LB* as the lowest solution cost in leaf search nodes. All the leaf search nodes that satisfy a near-optimal solution quality $c \leq w \cdot LB$ will be added to *Focal*. This new CBS will select a search node from *Focal* to expand, instead of that from *Open*. Compared to the original CBS, the new algorithm also performs the best-first search on *Focal*, but the search priority changes from the solution cost to a new heuristic function ψ. Typically, ψ is a handcrafted function that takes a solution as the input, and outputs a value that prefers solutions with fewer conflicts. We instead use a learned heuristic function based on the **Graph Transformer**.

Proposition. *Algorithm 5 is complete and bounded-suboptimal with a factor of* $w \geq 1$*, as mentioned in [9].*

Suppose the problem is feasible, but Algorithm 5 does not find a solution given a sufficient time budget. Then for an arbitrary search node N with a feasible solution, there exists an ancestor search node N^p added to *Open* but not expanded. Suppose N^{p*} is the search node with the lowest cost among these unexpanded ancestors. N^{p*} is not selected by *Focal*, since it is not expanded. Thus, either (i) N^{p*} is not in *Focal*, or (ii) N^{p*} is in *Focal* but not selected. (i) is impossible, because there do not exist infinitely many solutions that have costs lower than $\frac{1}{w} \cdot c(N^{p*})$. (ii) cannot happen, because there do not exist infinitely many solutions with costs lower than or equal to $w \cdot c(N^{p*})$. As a result, N^{p*} will be selected eventually and expanded. Therefore, we have proved the algorithm to be complete by contradiction. The focal search never expands search nodes with costs higher than w times the optimal solution; therefore, it is bounded-suboptimal

Algorithm 5 CBS with Biased Heuristics [9, 69]

Input: A MAPF instance and suboptimality factor w
Input: Heuristic function ψ (e.g., Graph Transformer)
Generate the root search node R with an initial solution
Initialize open list $Open \leftarrow \{R\}$
$LB \leftarrow c(R)$, and initialize focal list $Focal \leftarrow \{R\}$
while $Open$ is not empty
  $N^* \leftarrow \arg\min_{N \in Focal} \psi(\sigma(N))$
  $C \leftarrow$ first conflict in $\sigma(N^*)$
  if C does not exist
    return solution $\sigma(N^*)$
  Remove N^* from $Open$ and $Focal$
  if $\min_{N \in Open} c(N) > LB$
    $LB = \min_{N \in Open} c(N)$
    $Focal = \{N \in Open : c(N) \leq w \cdot LB\}$
  Generate two children nodes N_1 and N_2 from node N^*
  Add C_i to $\mathcal{T}(N_i)$, for $i = 1, 2$
  Call low-level planner to get $\sigma(N_i)$, for $i = 1, 2$
  Add N_i to $Open$, for $i = 1, 2$
  Add N_i to $Focal$ if $c(N_i) \leq w \cdot LB$, for $i = 1, 2$
return No solution

with a factor of w.

We note that the focal search described here is a special case of BCBS [9], i.e., BCBS $(w, 1)$, as the focal search is only applied to the high-level planner. In our graph-based problem settings, there is no significant improvement when applying the focal search to the low-level planner. Rather, if we introduce the focal search to the low-level planner, it would consume a notable portion of computation on the collision checking of edges, which has no improvement in the overall performance. We refer readers to Question 4 in Section 3.5.2 for further details.

3.4 Graph Transformers as Heuristic Functions

In this section, we describe the architecture of the Graph Transformer and how to train it represent a heuristic function that accelerates CBS.

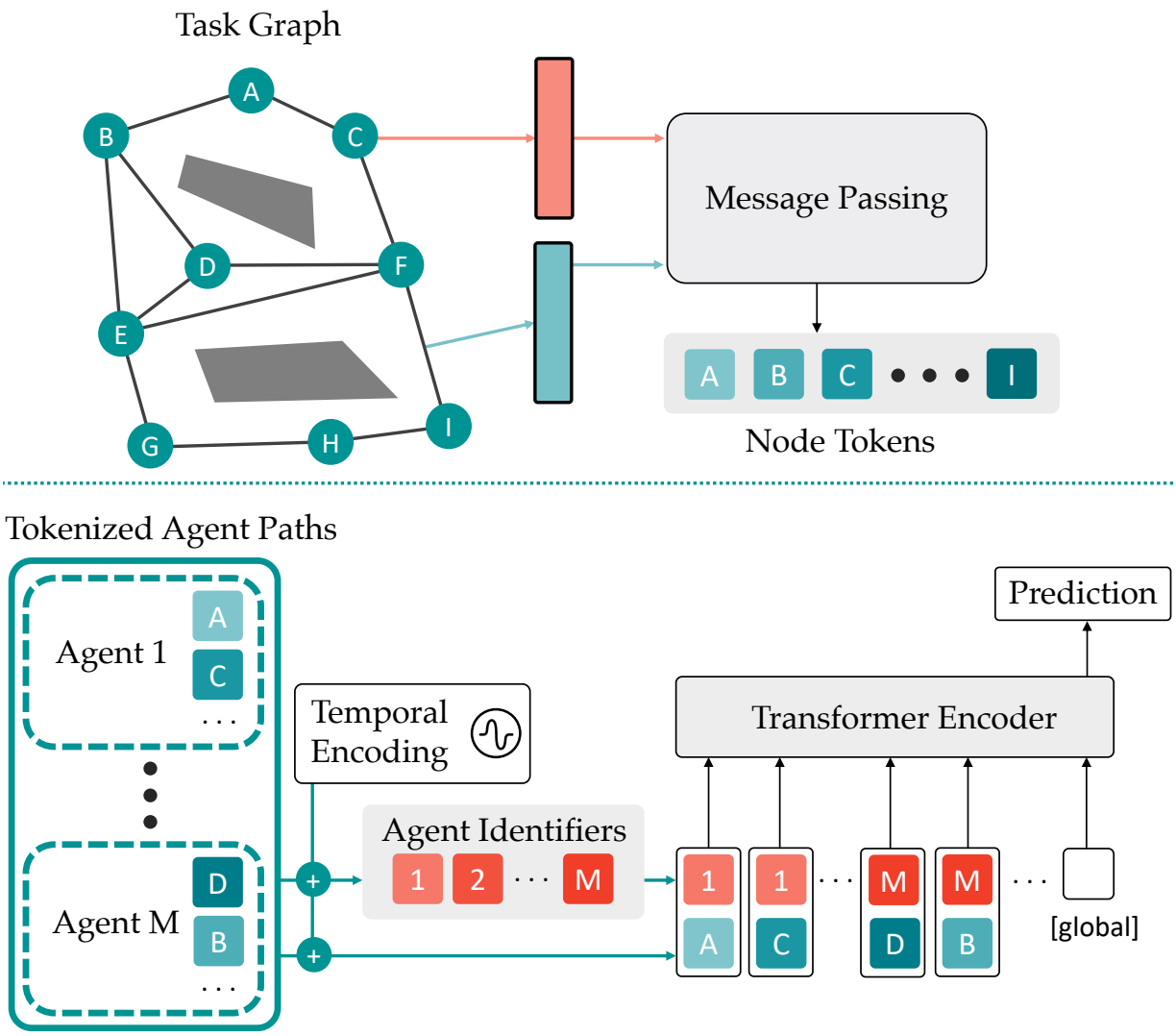


Figure 3.2: The proposed Graph Transformer architecture. It has several desired properties that are specifically designed to deal with MAPF inputs. See Section 3.4.1 for more details.

3.4.1 Network Architecture

The input to the graph transformer ϕ is a graph G, and a solution $\sigma = \{\sigma_i\}_{i \in [1 \cdots M]}$. The output $\phi(G, \sigma)$ predicts a scalar value as the heuristic. Such a predicted value correlates with the chance that the current search node will yield descendant search nodes with feasible solutions. For example, compared to those nodes that cannot eventually reduce the conflicts, the promising nodes leading to feasible solutions should have lower $\phi(G, \sigma)$ values.

The graph transformer has two stages: *graph tokenization* and *attentive aggregation*. We

describe each stage as follows.

Stage 1: Graph Tokenization. The graph tokenization transforms each graph vertex into an embedding using a Graph Neural Network (GNN). Here we use Message-Passing Neural Networks [55] (MPNN) as the GNN architecture. The input to the MPNN is a graph $G = \langle V, E \rangle$, where the feature d_i^v for each graph vertex $v_i \in V$ is its respective 2D position, and the feature d_l^e for each edge $e_l = (v_i \to v_j)$ is the relative position of v_j to v_i. With two linear layers f_x and f_y, the vertices and edges are first encoded as x and y using $\forall v_i \in V, x_i = f_x(d_i^v); \forall e_l \in E, y_l = f_y(d_l^e)$. Then, using three MLPs $\{f_k\}_{k\in[1,2,3]}$, the MPNN updates the information for each graph vertex $v_i \in V$ as follows:

$$x_i \leftarrow x_i + \max\{f_k(x_i, x_j, y_l)\}, \forall e_l : (v_i \to v_j) \in E. \tag{3.1}$$

After all x_i are updated using MLP f_1, the MPNN continues to update x_i using MLP f_2 and so on. The max denotes the max-pooling over the feature dimension. We use max-pooling to take a set with an arbitrary number of elements while ensuring robustness [130]. Since it is invariant to the permutation of these elements, the MPNN here can take graphs with an arbitrary number of vertices and edges, but also is permutation invariant by construction.

Stage 2: Attentive Aggregation. After we compute the token x_i for each graph vertex v_i from Stage 1, we model the inter-agent interactions using the Transformer [168]. The path of each agent σ_i is first tokenized as $\rho_i = \{x_j, \forall v_j \in \sigma_i\}$. To inject the temporal information of these tokenized solutions, we introduce Temporal Encoding [189]. The approach is similar to [168, 116] (as positional encoding in their settings). We denote $\rho_i^t \in \mathbb{R}^D$ as the vertex token of agent i at time step t. For each token ρ_i^t, we add it element-wisely with a temporal encoding

$\rho_i^t \leftarrow \rho_i^t + TE(t) \in \mathbb{R}^D$. The $2k$-th and $2k$+1-th dimensions of $TE(t)$ are as follows:

$$TE(t)_{2k} = \sin(t/10000^{2k/D}), \tag{3.2}$$

$$TE(t)_{2k+1} = \cos(t/10000^{2k/D}). \tag{3.3}$$

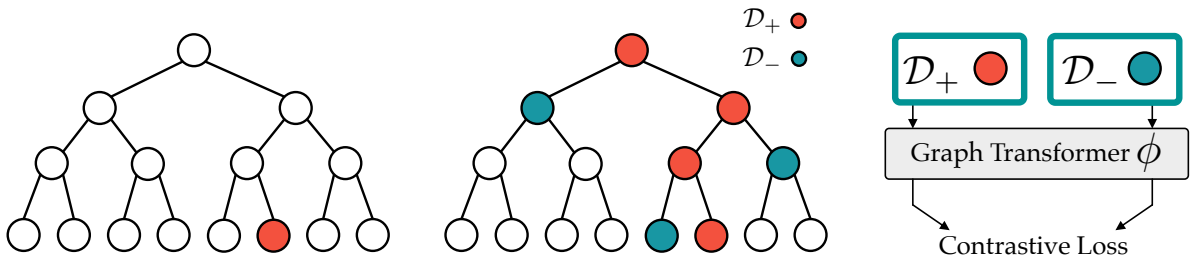
The hyperparameter 10000 is used following the common practice [168]. To encourage the model to be aware of which agent each token ρ_i^t belongs to, we concatenate the Agent Identifier τ_i to each token $\rho_i^t \leftarrow \rho_i^t || \tau_i$, similar to [87]. For each agent i, the agent identifier τ_i is calculated by taking the max-pooling over all its vertex tokens: $\tau_i = \max\{\rho_i^t\}, \forall t \in [1 \cdots T_i]$. Then, all tokens from the solution of all agents $\{\rho_i^t : \forall i \in [1 \cdots M], \forall t \in [1 \cdots T_i]\}$ will be fed as the input to the Transformer Encoder. For global prediction, we append an extra trainable token *global* to the input, following the common practice [35, 39]. The Transformer Encoder predicts an output for each input token, and we use the output of the trainable *global* token as ζ. With a linear layer $f_\phi : \mathbb{R}^D \to \mathbb{R}$, the final output is computed as $\phi(G, \sigma) = f_\phi(\zeta)$.

Here we use the Transformer Encoder, since it enables the Graph Transformer to take a variable number of tokens and model their dependencies, while preserving the invariance to the permutation of tokens. We refer readers to [168] for more details on the Transformer Encoder.

Properties of the Graph Transformer. By construction, the Graph Transformer is able to handle the input graph with a variable number of vertices and edges, agents with a variable total number, and the input solution with a variable length. Additionally, it is aware of the temporal information, and inter-agent interactions. Its output is permutation invariant to both the orders of graph vertices and the agents.

3.4.2 Training Graph Transformers

Data Generation. Given a MAPF instance, we first use CBS to generate feasible solutions. No data will be collected if CBS fails to solve the instance. If it succeeds, we start to collect positive



a. CBS Finds a Feasible Solution b. Labelling Samples c. Contrastive Training

Figure 3.3: The training framework for Graph Transformer. We use a supervised Contrastive Loss. The labels are generated from the CBS search tree.

and negative samples from its search tree. A positive sample will be collected by dataset $\mathcal{D}_+$, if itself or one of its descendant search nodes contains the feasible solution. A negative sample will be collected by dataset $\mathcal{D}_-$, if it is a sibling search node of a positive sample. We record each sample's graph G and solution σ. In addition, we record the value d as its respective depth in the search tree.

Supervised Contrastive Learning. The objective of the model is to learn a ranking of the samples. Namely, given an arbitrary pair of positive sample (G, σ_+, d_+) and a negative sample (G, σ_-, d_-) from the same MAPF graph G, we learn the following ranking:

$$\phi(G, \sigma_+) < \phi(G, \sigma_-), \text{ if } d_+ \geq d_-.$$

We illustrate the intuition here. Imagine such ranking is learned perfectly and $w = \infty$, meaning all the leaf search nodes will be in *Focal*. Suppose at some time point, *Focal* includes 1 positive sample p_+. *Focal* may or may not include negative samples. If they exist, then their depths are no deeper than $d(p_+)$. Define this condition as a loop invariant. Then p_+ will be selected and expanded first according to the ranking. If p_+ is the feasible solution, then the algorithm terminates. Otherwise, *Focal* will have one positive sample p'_+ and some negative samples, and all negative samples have depths no deeper than $d(p'_+)$. Thus, the loop invariant remains true. The loop invariant is also true for the base case, where *Focal* only has the root search node. Therefore, by learning such ranking, we encourage Algorithm 5 to expand positive

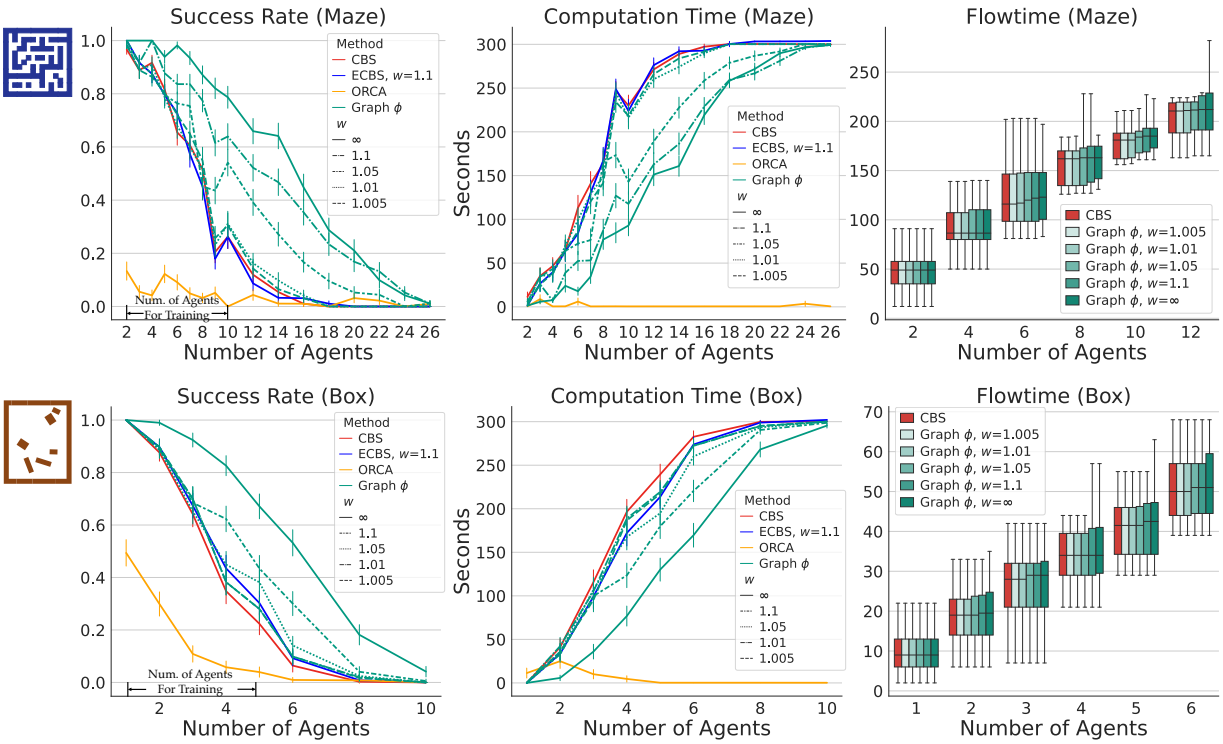


Figure 3.4: Success rates, computation time, and flowtime within the runtime limit of 5 minutes, as functions of the number of agents. The results are averaged over 100 test instances for each setting of the agent number. We evaluate our approach with $w \in [1.005, 1.01, 1.05, 1.1, \infty]$, and compare its performance with CBS, ECBS ($w = 1.1$) and ORCA. Though trained with relatively few agents, results have shown that our approach generalizes well and significantly outperforms the baselines.

samples first and expand the negative samples as few as possible, which could save significant computation and greatly accelerate CBS.

We use supervised contrastive learning to train a Graph Transformer ϕ that ranks the positive samples above the negative samples. Given an arbitrary pair of positive and negative samples, $p_+ : (G_+, \sigma_+, d_+) \in \mathcal{D}_+, p_- : (G_-, \sigma_-, d_-) \in \mathcal{D}_-$, this pair is defined to be valid as: $\mathbb{I}(p_+, p_-) : (G_+ = G_-) \wedge (d_+ \geq d_-)$. With a hyperparameter $\gamma = 0.1$, we define $\delta(x) : \max(0, \gamma + x)$. We aim to minimize the Contrastive Loss as follows:

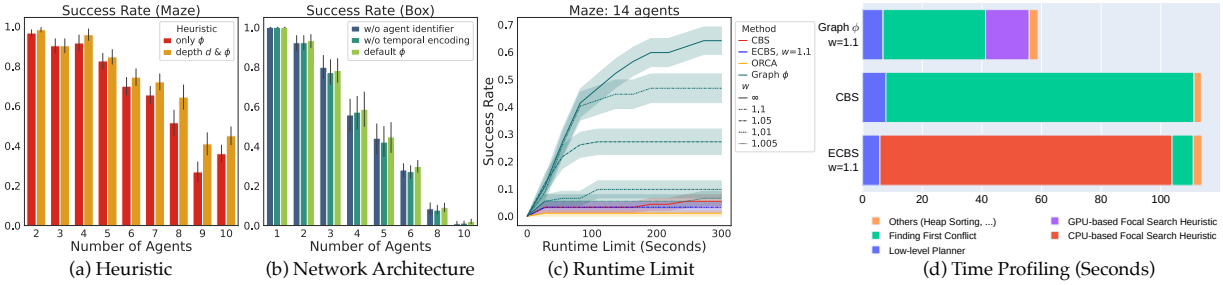
$$\frac{1}{L} \sum_{\substack{p_+ \in \mathcal{D}_+ \\ p_- \in \mathcal{D}_-}} \delta(\phi(G_+, \sigma_+) - \phi(G_-, \sigma_-)) \cdot \mathbb{I}(p_+, p_-), \tag{3.4}$$

where $L = \sum_{p_- \in \mathcal{D}_-}^{p_+ \in \mathcal{D}_+} \mathbb{I}(p_+, p_-)$ is the total number of valid sample pairs, and $\mathcal{D}^+, \mathcal{D}^-$ are the datasets of positive and negatives samples respectively.

Once the training of Graph Transformer ϕ reaches convergence, we could deploy it as the heuristic function ψ in Algorithm 5. However, in practice, we found that the model would predict relatively low values for multiple search nodes in the focal set, indicating that all of them may lead to feasible solutions. To break the tie, we instead represent ψ as the depth d combined with ϕ, i.e., $\psi = \langle -d, \phi \rangle$. It means that the algorithm would first prefer the search nodes with deeper depths; if there exist multiple search nodes with the deepest depths, then it would prefer the search nodes with lower ϕ. Such a design enables the algorithm to make decisions consistently if multiple promising nodes exist. Without further specification, we denote our approach with such a heuristic as **Graph** ϕ.

3.5 Experiments

3.5.1 Main Experiments



(a) Heuristic (b) Network Architecture (c) Runtime Limit (d) Time Profiling (Seconds)

Figure 3.5: We conduct 4 various ablation studies to evaluate the proposed method systematically. See Section 3.5.2 for more details.

Experimental Setup. We design two types of environments for evaluation: Maze and Box (see Fig. 3.1). Each environment includes 2700 MAPF instances with samples generated by CBS for training. For testing, there are 100 MAPF test instances w.r.t. each agent number setting. We generate a random map for each Maze instance and a set of random obstacles for each Box

instance. The graph and start and goal vertices are also generated randomly for each instance.

The training instances vary between 2 and 10 agents for Maze, and vary between 1 and 5 agents for Box. The test instances vary between 2 and 26 agents (2-10, 12, 14, 16, 18, 20, 22, 24, 26) for Maze, and vary between 1 and 10 agents (1-5, 6, 8, 10) for Box. We ensure that the test instances are unseen in the training set. All experiments were conducted using a 12-core, 3.2Ghz i7-8700 CPU and 4 Nvidia GTX 1080Ti GPUs. We test $w \in [1.005, 1.01, 1.05, 1.1, \infty]$ for our method. For all methods, we set the runtime limit as 5 minutes, following the common practice [69].

Baselines. We compare our method with 3 baselines: (i) Conflict-Based Search. (ii) Enhanced Conflict-Based Search (ECBS): a bounded-suboptimal version of CBS [9]. It applies focal search to both high-level and low-level planners, using hand-crafted heuristic functions. We choose its $w = 1.1$. (iii) ORCA [166]: a reactive collision avoidance algorithm, which works effectively in low density environments.

We find that there are very few open-source implementations that could directly apply CBS and ECBS to non-grid-based problems. As a result, we implement all tree-search methods (CBS, ECBS, and our method) from scratch using Python. We use basically the same framework when implementing CBS, ECBS, and our approach. To make the comparison fair, we ensure that all these 3 methods are aggressively optimized by strictly following the C++ implementations [1] and original paper [151, 9], and using Bayesian hyperparameter search for the w of ECBS.

Evaluation Metrics. Our evaluation includes 3 metrics: 1) *Success Rate*, the ratio of the number of successful instances to the total number of test instances. An instance is successful if all agents reach their goals with no collision before the timeout happens. 2) *Computation Time*, the average computational time for each instance, including the failed ones. [2] 3) *Flowtime*, the sum of all the agents' travel time. We only compare the flowtime of the CBS to our method based

[1] https://github.com/whoenig/libMultiRobotPlanning/

[2] Since CBS and ECBS call different low-level planners (A* and A*-ε), we consider the computation time to be the fairest metric to evaluate the efficiency, instead of counting the number of expanded nodes, for instance.

on the instances where both methods succeed, to show our approach's bounded-suboptimality guarantees.

Overall Performance. We demonstrate the overall performance in Figure 4.7. Our method significantly outperforms the three baselines in both Maze and Box. It requires much less computation time and achieves significantly higher success rates. Furthermore, though only trained with relatively few numbers of agents, our method generalizes remarkably well to a higher number of agents. For example, our network trained by 2 to 10 agents can be generalized up to 26 agents in Maze, while our policy trained by 1 to 5 agents can be generalized up to 10 agents in Box. In particular, with $w = 1.1$, our method achieves the success rates of $47\%, 23\%, 10\%, 1\%$ in Maze with 14, 18, 22, and 26 agents, and achieves the success rates of $62\%, 30\%.4\%, 0.5\%$ in Box with 4, 6, 8, and 10 agents. On the other side, CBS fails to solve Maze with 18 agents and Box with 10 agents within the timeout. Similarly, ECBS starts to fail in all instances with 20 agents for Maze and with 10 agents for Box. In addition, since the bounded-suboptimality of our algorithm is proved theoretically, it is not surprising to see that the solution qualities (flowtime) of our method are very close to the optimal solutions. Finally, ORCA easily fails in highly dense environments, and such performance is consistent with previous works [3].

3.5.2 Ablation Study

In this section, we investigate four questions:

(a) Is incorporating depth information into the heuristic beneficial to the performance? Fig. 3.5 (a) illustrates the comparison between the heuristic with only ϕ, and with depth d and ϕ, on tests from 2 to 10 agents in Maze. The result shows the effectiveness of introducing depth information. The improvement becomes more noteworthy as the number of agents grows, which validates our choice.

(b) Do the Temporal Encoding and the Agent Identifier improve the performance? Fig. 3.5 (b) demonstrates the performances in Box with and without the Temporal Encoding and

Agent Identifier. The results show that Agent Identifier and Temporal Encoding improve the success rate. The average improvement of introducing Agent Identifier and Temporal Encoding over the non-default settings is 16±33%.

(c) How will the runtime limit affect the performance? We take the tests with 14 agents in Maze as an example. We set different runtime limits from 25 seconds to 300 seconds with the interval as 25 seconds. In Fig. 3.5 (c), we show that our methods outperform the baselines regardless of how the runtime limit changes. When the limit is 300 seconds, our method achieves a success rate of 64% ($w = \infty$), while CBS, ECBS, and ORCA only have 5%, 3%, and 1% respectively.

(d) Time profiling each module of the planners. Finally, we wish to answer the question of why the performances of ECBS considerably degrade once we apply it to non-grid-based MAPF instances, compared to the traditional grid-based settings. We profile each method, and average the results over the Maze tests with 2-10 agents.

Fig. 3.5 (d) illustrates that ECBS spends considerable computation on the focal heuristic calculation. Such behavior is reasonable, since now for dense graphs, the heuristic is calculated by checking the collisions along edges. Such collision checking is often the main bottleneck for planning and by itself NP-hard in general [23, 75]. Meanwhile, our method uses a learned function for the focal heuristic calculation (the GPU part), which only takes 15% computation cost compared with ECBS. Compared with CBS and ECBS, our method only requires 50% of the total computation time.

3.6 Conclusion and Future Work

We proposed the Graph Transformer as a heuristic function to accelerate CBS by guiding the tree search towards promising nodes with feasible solutions. While achieving significant acceleration, our approach guarantees provable completeness and bounded-suboptimality. We

show that in two continuous environments with dense motion graphs, our method outperforms three classical MAPF baselines (CBS, ECBS, and ORCA), while generalizing to unseen tests with a higher number of agents remarkably well. Future works include real-world experiments and improving the method for 10x-100x agents.

3.7 Acknowledgments

Chapter 3, in full, is a reprint of Chenning Yu, Qingbiao Li (co-first authors), Sicun Gao, Amanda Prorok, "Accelerating Multi-Agent Planning Using Graph Transformers with Bounded Suboptimality", *ICRA, 2023*. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Learning Control Admissibility Models with Graph Neural Networks for Multi-Agent Navigation

Deep reinforcement learning in continuous domains focuses on learning control policies that map states to distributions over actions that ideally concentrate on the optimal choices in each step. In multi-agent navigation problems, the optimal actions depend heavily on the agents' density. Their interaction patterns grow exponentially with respect to such density, making it hard for learning-based methods to generalize. We propose to switch the learning objectives from predicting the optimal actions to predicting sets of admissible actions, which we call control admissibility models (CAMs), such that they can be easily composed and used for online inference for an arbitrary number of agents. We design CAMs using graph neural networks and develop training methods that optimize the CAMs in the standard model-free setting, with the additional benefit of eliminating the need for reward engineering typically required to balance collision avoidance and goal-reaching requirements. We evaluate the proposed approach in multi-agent navigation environments. We show that the CAM models can be trained in environments with

only a few agents and be easily composed for deployment in dense environments with hundreds of agents, achieving better performance than state-of-the-art methods.

4.1 Introduction

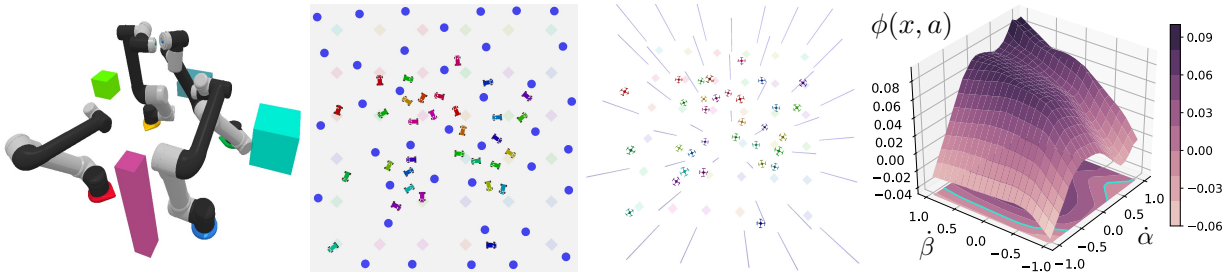
Multi-agent navigation is a longstanding problem with a wide range of practical applications such as in manufacturing [8, 190], transportation [185], and surveillance [119]. The goal is to control many agents to all achieve their goals while avoiding collisions and deadlocks. Such problems are multi-objective in nature, and the control methods depend heavily on the density of the agents, with the computational complexity growing exponentially in such density [124, 20, 125, 17].

Recently, reinforcement learning approaches have shown promising results on multi-agent navigation problems [50, 84, 104, 107, 136], but there are still two well-known sources of the difficulty. First, balancing safety and goal-reaching requirements often lead to ad hoc reward engineering that is highly dependent on the environments and the density of the agents [2, 57]. Second, it is shown that such neural control policies leaned on RL approaches are hard to generalize when the agent density in the environment changes, which leads to distribution drifts. The core difficulty for generalizability is that the RL approaches are designed to capture the optimal actions specific to the training environments and rewards. Nevertheless, such optimal actions may change rapidly when the density and interaction patterns change during deployment. The unnecessary optimization of control policies in the training environments makes it fundamentally hard to transfer the learned results to new environments and achieve compositionality in the deployment to varying numbers of agents.

We propose a new learning-based approach to multi-agent navigation that shifts the focus from learning the optimal control policy to a set-theoretic representation of *admissible* control policies to achieve compositional inference. By decoupling the multi-agent navigation tasks,

we avoid the ad-hoc reward engineering using a simple goal-reaching preference function and a learnable set-theoretic component for collision avoidance. We use graph neural networks (GNNs) to represent the set of admissible control actions at each state as Control Admissibility Models (CAM), which are trained with a small number of agents during training with sparse rewards. In online inference, we compose the CAMs and apply goal-reaching preference functions to infer the specific action to take for each agent. We show that CAMs can be learned purely from the data collected online in the standard RL setting, with no expert or human guidance. We propose relabelling through backpropagation (Section 4.3.3) for the CAMs to learn rich information from the transitions even given sparse rewards. As shown in Figure 4.1(d), the learned models of the admissible actions are typically complex and suitable for GNN representations. Moreover, in online inference, we decompose the state graph of the agent into much smaller subgraphs and aggregate the CAMs from them. Such compositionality allows us to train CAMs with only a small number of agents, directly deploy them in much denser environments, and achieve generalizable learning and inference.

We evaluate the proposed methods in various challenging multi-agent environments, including robot arms, cars, and quadcopters. Experiments show that the CAM generalizes very well. While the training environments only have at most 3 agents, the learned CAM models can be directly deployed to hundreds of agents in comparable environments and achieve a very high success rate and low collision rate. We analyze the benefits of the proposed methods compared to state-of-the-art approaches. Furthermore, we show a zero-shot transfer to a multi-agent chasing game, demonstrating that the trained CAM generalizes to other tasks that are not limited to navigation.



(a) UR5 (b) Car (c) Drone (d) Landscape of CAM

Figure 4.1: **(a-c)** Illustrations of the multi-agent environments. We show videos in the supplemental materials. **(d)** An example landscape of the proposed CAM ϕ in the Drone environment. The axes are two dimensions in the action space, corresponding to the rotation rates in pitch and roll. We project the landscape into a 2D plane at the bottom. The blue line on the plane denotes the zero level set, which divides the action space into the admissible and inadmissible sets.

4.2 Related Work

Multi-Agent Navigation. Classical Multi-Agent Path Finding is one of the most popular definitions of the multi-agent navigation task, where the state space lies in a pre-constructed graph shared among agents [157]. Under this setting, heuristic-based methods have been used to generate high-quality solutions, such as Priority-Based Search and Conflict-Based Search [110, 151]. Learning-based methods have also been proposed to accelerate the planning time, using CNN and GNN [101, 102, 145]. In this work, we focus more on the continuous state space with reactive planning. Traditional methods, including Optimal Reciprocal Collision Avoidance [14], Buffered Voronoi Cells [5], and Artificial Potential Functions [85, 10, 52, 63], are designed manually to address specific structural properties. On the other hand, learning-based multi-agent methods using reinforcement learning and imitation learning [50, 84, 104, 107, 136, 140, 152], may or may not generalize when there is covariant shift for test tasks. Recently, MACBF [132] incorporates multi-agent safe certificates [1, 59, 172] into a learning-based framework, which shows the generalization capability while preserving the safety properties. Nevertheless, in [132] the learning of the neural controller requires a reference controller, which may constrain the potential of the neural controller, when the actions dissimilar to the reference actions could yield better rewards and safer results. Though learned in an online fashion, our work only assumes

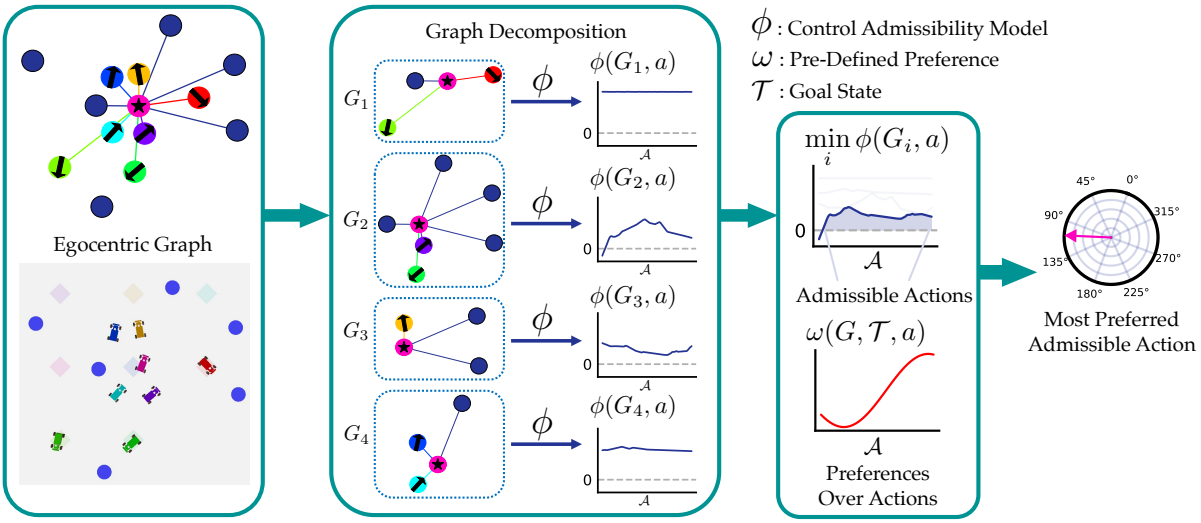


Figure 4.2: An overview of the proposed CAM approach. At each time step, the state of each agent is an egocentric graph. The graph is further decomposed at inference time into a set of subgraphs to follow the training data distribution. The CAM takes these subgraphs and outputs the admissibility scores along with a set of sampled candidate actions. We filter out the inadmissible actions by checking whether the minimum of the scores is below 0. Given a predefined preference over actions, the model outputs the most preferred action in the admissible set for each agent.

sparse reward and avoids the reward hacking problem.

Graph Neural Networks. Graph Neural Networks (GNN), including Graph Convolutional Networks [88], Message Passing Neural Networks [55], Interaction Networks [12], are deep neural networks that operate on graph-structured data [147]. Graph neural networks are permutation-equivariant to the orders of nodes on graph [186, 130], which is important for homogeneous multi-agent problems since the order among agents should not matter. In robotics, the effectiveness of GNN has been shown on tasks such as path planning [101, 102], motion planning [83, 182], coverage and exploration [192, 163], and SLAM [144]. In this work, we use GNN as an architecture to implement CAM due to the multi-agent navigation problem can be modeled as a graph-based problem in nature. However, we believe that the proposed CAM could generalize to other inputs that are not graphs.

4.3 Control Admissibility Models

We present Control Admissibility Models (CAMs), a general representation of the feasible control set. CAM is task-compositional, robust to reward design, and can be trained with an online pipeline similar to model-free RL. With graph decomposition, CAM can plan a high-quality solution for a large number of robots, which could be significantly out of the distribution of the training tasks.

4.3.1 Control Admissibility Models (CAMs)

We define CAMs as a general representation of the feasible actions. In general, CAMs are function approximators that behave as the characteristic functions of such sets. By evaluating the CAM values we can efficiently determine whether an action is feasible. Formally,

Definition 1 (Control Admissibility Models and Admissible Set)**.** *Given CAM* ϕ*, state* x*, action space* $\mathcal{A}$*, the admissible set is defined as* $\Delta(\phi, x) : \{a \mid a \in \mathcal{A}, \phi(x,a) \geq 0\}$.

Compositionality of CAMs. An important feature of CAMs is that they can be composed together. Given two CAMs ϕ_1 and ϕ_2, we want to find the admissible actions that satisfy both these two CAMs:

$$\{a \mid \phi_1(x,a) \geq 0\} \cap \{a \mid \phi_2(x,a) \geq 0\}$$

Proposition 1 (Composition)**.** *If* ϕ_1 *and* ϕ_2 *are CAMs, then* $\Delta(\phi_3, x)$ *defines* $\Delta(\phi_1, x) \cap \Delta(\phi_2, x)$*, given* $\phi_3 = \min\{\phi_1, \phi_2\}$.

4.3.2 Graph Neural Networks for CAMs

Though CAM can be applied to various single-agent problems (see Section 4.4.1 as an example), we focus on multi-agent problems in this paper. At each timestep, the state for each agent is an egocentric graph $G = \langle V, E \rangle$, where V are the vertices and E are the edges. Vertices

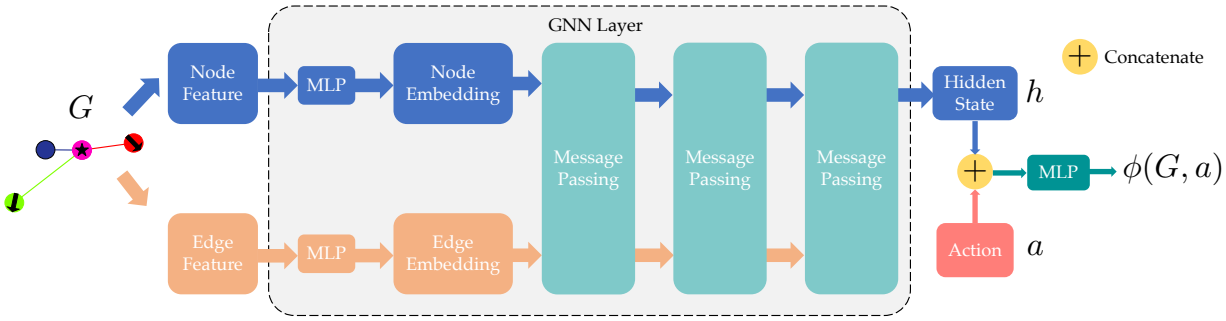


Figure 4.3: The GNN architecture of the proposed CAM. A state for each agent is an egocentric graph G, which includes the node features and edge features. The CAM first encodes the graph using the GNN layer. The GNN layer then outputs the hidden state h for the current agent based on the updated embedding of the corresponding node. Given action a and another MLP f, the CAM generates the admissibility score using $\phi(G,a) = f(h,a)$.

V is composed of agent vertices V_a and static obstacle vertices V_o. Edges E are connecting all the neighbor vertices to the current agent, i.e. $E = \{(v_a^j, v_a^i)|a_j \in \mathcal{E}(a_i) \cup \{(v_o^j, v_a^i)|o_j \in \mathcal{E}(a_i)\}$, where $\mathcal{E}$ denotes the neighbor vertices. The one-hot vector is used as the features of V, denoting the types of vertices. For the features of edges E, we use relative positions and the states of the two connected vertices, along with a one-hot vector to indicate the edge types. Zero paddings are adopted if necessary [91].

The CAM ϕ has two components: The first component is a GNN layer, which transforms the graph G to a hidden state vector h for each agent. The second component is a fully-connected layer f, which takes the hidden state h and an action a, and predicts the admissibility score $\phi(G,a) = f(h,a)$. Such a design enables fast computation of the admissibility scores for all the possible state-action pairs. We can merge the computation of the hidden state for all state-action pairs which share the same input graph into one forward passing in the GNN layer.

We describe the GNN layer with the following details. The vertices and the edges are first embedded into latent space as $m^{(0)} = g_m(v), n^{(0)} = g_n(e)$, given fully-connected layers g_m, g_n. Taking m, n, the GNN aggregates the local information for each vertex from the neighbors through K GNN layers. For the k-th layer, it performs the message passing with 2 fully-connected layers

$f_m^{(k)}$ and $f_n^{(k)}$:

$$
\begin{aligned}
m_i^{(k+1)} &= m_i^{(k)} + \max\{f_m^{(k)}(n_l^{(k)}) \mid e_l : (v_j, v_i) \in E\}, \forall v_i \in V \\
n_l^{(k+1)} &= n_l^{(k)} + f_n^{(k)}(m_i^{(k+1)}, n_l^{(k)}), \forall e_l : (v_j, v_i) \in E
\end{aligned}
\tag{4.1}
$$

We assign the hidden state h with the final node embedding $m_i^{(K)}$, where i corresponds to the current agent. We use max as the aggregation operator to gather the local geometric information due to its empirical robustness to achieve the order invariance [132, 130]. We use the residual connection to update the vertex embedding and edge embedding due to its simplicity and robust performance for deep layers [61, 67]. Moreover, each agent's hidden state h is only updated based on its egocentric graph, which is entirely invariant to the other agents' hidden states. This architecture decentralizes the decision-making of the whole swarm.

4.3.3 Training CAMs in Reinforcement Learning

We train CAMs in the same online setting as model-free reinforcement learning procedures. At each time step, the agent perceives the observation from the environment and takes action. The state-action pairs for every transition are labeled as safe or unsafe and then appended to the replay buffer. The CAM is updated every time a certain number of transitions are collected. The main differences between our training approach and the standard RL methods are three-fold: **(i)** instead of using an actor-network, the agent chooses the action among the sampled actions based on the admissibility score generated by CAM. **(ii)** the label is binary for each state-action pair instead of calculating the cumulative future rewards. **(iii)** the training objective is non-bootstrapped, which avoids the overestimation issue and is more stable. We design these three improvements around the underlying structure of the admissibility problem. We provide further explanations as follows.

Sparse Reward Setting. We model the problem as the standard Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$ [160]. S is the state space represented as graphs G. The action space $\mathcal{A}$ is

Algorithm 6 CAM Algorithm for Training

Input: Preference function ω, exploration noise $\mathcal{N}$
Initialize CAM ϕ
Initialize replay buffer R
for episode = 1, M **do**
  for t = 1, T **do**
    Sample candidate actions $\{a\} \subseteq \mathcal{A}$
    $\{G\}, \{\mathcal{T}\} \leftarrow$ all agents' current states and goal states
    Select action for each agent according to $\phi(G,a)$, $\omega(G,\mathcal{T},a)$, and $\mathcal{N}$
    Execute the selected actions and observe the next states s_{t+1}
    Label the transition (s_t, a_t, s_{t+1}) with y_t by checking whether s_{t+1} is in danger set
    Store transition (s_t, a_t, y_t, s_{t+1}) in R
  Relabel R through backpropagation
  Update ϕ using Equation 4.2

continuous. The reward r is sparse: it only returns non-zero values when the agent enters the region to avoid and the region to reach. These certain regions can also be dynamic, i.e., danger regions centering around other agents.

Preference Function. Given current state G and goal state $\mathcal{T}$, we assume a preference function over actions, i.e., $\omega(G,\mathcal{T},\cdot)$, is given. The preference function solely focuses on the goal-reaching. For instance, it can simply be the L2 distance from the next state to the goal, since CAM will achieve the obstacle avoidance. In the experiments, we try to choose the preference functions as simple as possible.

Action Selection via Sampling. At every timestep, the agent chooses an action among a batch of actions sampled from the action space. Given a state x, the admissibility value every action a is computed with $\phi(x,a)$. If there exist one or multiple actions that satisfy $\{a \mid \phi(x,a) \geq 0\}$, given the goal state $\mathcal{T}$ and a preference $\omega(x,\mathcal{T},\cdot)$ over actions, we choose the action that has the highest preference score in the admissible set. Namely, $\arg\max_a \{\omega(x,\mathcal{T},a) | \phi(x,a) \geq 0\}$. If there does not exist any admissible action, in order to fulfill the admissibility property maximally, the agent chooses the action that has the highest admissibility score, $\arg\max_a \{\phi(x,a)\} + \mathcal{N}$, where $\mathcal{N}$ is an exploration noise. We model the exploration noise under the uniform distribution in our

experiment.

Training Loss of CAM. Given admissible transitions $R_0 \subseteq \mathcal{S} \times \mathcal{A}$ and inadmissible transitions $R_d \subseteq \mathcal{S} \times \mathcal{A}$, we train the CAM with the following objective:

$$L_B = \frac{1}{|R_0|} \sum_{(x,a)\in R_0} \text{ReLU}(\gamma_1 - \phi(x,a)) + \frac{1}{|R_d|} \sum_{(x,a)\in R_d} \text{ReLU}(\gamma_2 + \phi(x,a)) + \frac{1}{|R_0|} \sum_{(x,a)\in R_0} \text{ReLU}(\gamma_3 - \dot{\phi}(x,a) - \lambda\phi(x,a)) \quad (4.2)$$

where $\gamma_1, \gamma_2, \gamma_3 \geq 0$ are the coefficients to enlarge the margin of the learned boundary. Instead of simply classifying the transitions with the first and the second loss terms, we encourage the forward invariance property by adding the third term, similar to Control Lyapunov Functions and Control Barrier Functions [1]. In practice, we approximate $\dot{\phi}(x,a)$ by $\phi(x',a') - \phi(x,a)$, where (x',a') is the next state-action pair on the trajectory. Intuitively, once all pairs of consecutive transitions satisfy this condition, the transitions would form a forward invariant set, and any trajectory starting from inside the invariant set will never cross the admissible boundary. The $\lambda\phi(x,a)$ term encourages the trajectory to be asymptotically admissible, even if the agent enters the inadmissible region sometimes.

Relabeling Through Backpropagation. We first label the collected transitions as admissible and inadmissible based on whether the next state is in the danger region. Once the whole trajectory terminates, we relabel these transitions through backpropagation. Intuitively, suppose the agent encounters the collision along the trajectory. In that case, it means that starting from some timestep, it enters the inadmissible region and fails to get back to the admissible set (similar to Intermediate Value Theorem [142]). Without relabeling, the labels of the transitions from this step to the collision event will be admissible, which is incorrect. To solve this issue, we relabel these critical transitions.

A state-action pair (x,a) on the trajectory is relabelled as inadmissible, if **(i)** the next state-action pair (x',a') is inadmissible, and **(ii)** no admissible action is found at state x' - namely,

$\max_{a'} \phi(x', a') < 0$. Intuitively, the relabelling propagates the inadmissibility from where the agent enters the danger region back to a state, where the agent can make a decision leading to admissible regions. The first condition ensures the optimistic behavior of the CAM. We only relabel those transitions which stay in the inadmissible region and encounter danger in the future. Such optimism is important when no feasible trajectory exists at the early training stage. It encourages the agent to explore instead of predicting almost every transition to be inadmissible. The second condition ensures that there exist no admissible actions approximately by inspecting the admissibility score for all the actions. Furthermore, for the transitions that have potential admissible actions, they will not be relabelled as inadmissible easily at the early training stage since $\max_{a'} \phi(x', a') < 0$ is hard to meet for a moderately initialized CAM.

4.3.4 Online Inference with Graph Decomposition

At inference time, the agent behaves in the same manner as the training stage, except that there is no exploration noise during inference. When the CAM is applied directly to a large swarm of agents, however, we find that the performance degrades significantly due to the distribution shift. Such degradation occurs considerably when the inference task has a much higher density of agents. To solve this issue, we leverage the compositional property of CAMs and propose graph decomposition.

Decomposition on Large Graphs. We can break down the whole graph G into subgraphs $\{G_k\}$, where these subgraphs satisfy the training distribution, e.g., the number of agent vertices does not exceed a certain value. We construct the subgraphs repeatedly until all the edges in G appear at least once in the subgraphs. For each subgraph G_k, and action a, we calculate the admissibility score as $\phi(G_k, a)$. Though each subgraph defines a less strict subtask for the agent to perform, the admissible set for the original task should lie in the intersection of all these admissible sets of the subtasks. As a result, these values on the subgraphs can be composed to represent the admissibility score on the entire graph $\phi(G, a)$, which could be totally out of distribution. To

compose the admissibility values, we follow Proposition 1 and use $\phi(G,a) = \min_{G_k \subseteq G}\{\phi(G_k,a)\}$.

Algorithm 7 CAM Algorithm for Inference

Input: CAM ϕ, preference function ω, distribution P of training data
for t = 1, T **do**
  for each agent's current state G, goal state $\mathcal{T}$ **do**
    Sample candidate actions $\{a\} \subseteq \mathcal{A}$
    Initialize $\phi(G,a)$ as ∞ for all sampled candidates
    repeat
      Sample $G_i \subseteq G$, where $G_i \sim P$
      $\phi(G,a) = \min(\phi(G,a), \phi(G_i,a))$ for all candidates a
    until all edges on G are sampled at least once
    Select action for the agent according to $\phi(G,a)$ and $\omega(G,\mathcal{T},a)$
  Execute the selected actions and observe the next states s_{t+1}

4.4 Experiments

4.4.1 Proof of Concept: CAM for Single Agent Environment

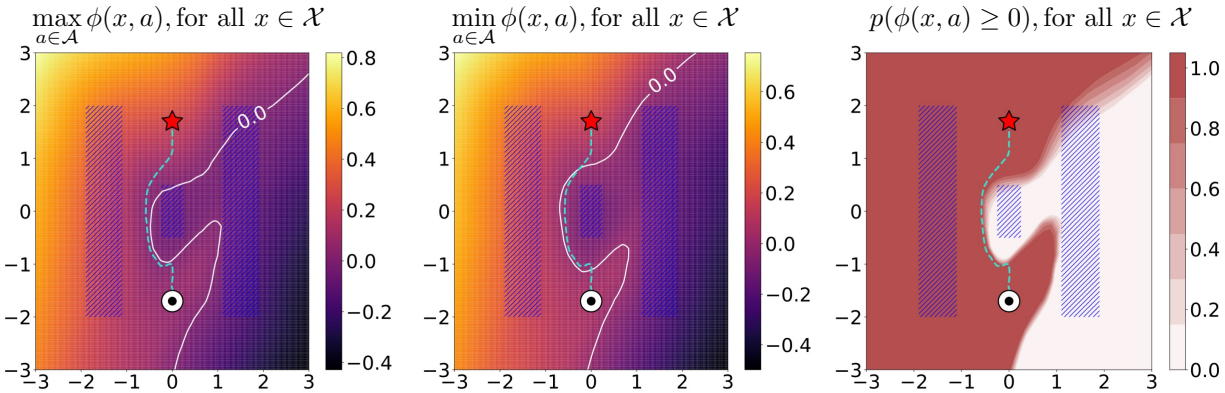


Figure 4.4: A proof of concept for the proposed CAM in a single-agent environment. The agent aims to reach the goal while avoiding obstacles. The black point denotes the starting state, and the red star denotes the goal state. The shaded blue regions denote the obstacles. Given the learned CAM ϕ, action space $\mathcal{A}: \{-0.1, 0.1\}^2$, and an arbitrary state $x \in \mathcal{X}: \{-3,3\}^2$, we illustrate $\max_{a\in\mathcal{A}} \phi(x,a)$, $\min_{a\in\mathcal{A}} \phi(x,a)$, and $p(\phi(x,a) \geq 0)$ respectively. We show that the learned CAM never leaves the admissible set, i.e. $\{x \in \mathcal{X} : \max_{a\in\mathcal{A}} \phi(x,a) \geq 0\}$, along the trajectory.

We begin by analyzing the CAM agent in a single-agent environment. We perform such

analysis using a 2D minimal example environment, where we place three danger regions to form narrow passages, as shown in Figure 4.4. We illustrate $\max_{a\in\mathcal{A}}\phi(x,a)$, $\min_{a\in\mathcal{A}}\phi(x,a)$, and $p(\phi(x,a)\geq 0)$ respectively. Intuitively, $\max_{a\in\mathcal{A}}\phi(x,a)$ indicates whether there exists any admissible actions that drive the agent to the admissible set. $\min_{a\in\mathcal{A}}\phi(x,a)$ indicates the possibility to enter the inadmissible sets if actions are picked randomly. As what we show in the figure, the trajectory (blue dashed line) does not intersect with the admissibility boundary, i.e. $\{x\in\mathcal{X}:\max_{a\in\mathcal{A}}\phi(x,a)=0\}$. This example stays consistent with the forward-invariance objective for learning the CAM: if the agent executes an admissible action $a\in\mathcal{A}:\phi(x,a)\geq 0$ at every time step, then it will never leave the admissible set.

We provide more details on the experiment setup as follows:

Dynamics. The agent here follows the single integrator and controls a 2D action. The state $x=[p_x,p_y]$, representing the position in the 2D space. The 2D action $a=[v_x,v_y]\in\mathcal{A}=[-1,1]^2$ is composed of the velocities along x and y axis, and the dynamics to compute the next state x' is

$$x'=f(x,a)=x+\begin{bmatrix}v_x\\v_y\end{bmatrix}\cdot dt \tag{4.3}$$

We choose $dt=0.05$ in our experiment. In this example, we use an MLP to represent CAM, since the state here is solely a vector.

Task Distribution. The goal state $[p_x=0,p_y=1.7]$, and the obstacles are all fixed during training and test. The starting state of the agent is fixed as $[p_x=0,p_y=-1.7]$.

Collision and Goal-reaching Criteria. We define the agent as a circle with a radius of 0.15. The goals are circles with a radius of 0.15. The agent is in a collision if it intersects with the obstacles. If the agent intersects with the 2D goal, then the goal is reached.

Preference Function ω. The preference function calculates the negative of distance to the goal based on the next state: $\omega(x,\mathcal{T},a)=-||(p'_x,p'_y)-\mathcal{T}||_2^2,[p'_x,p'_y]=x'=f(x,a)$.

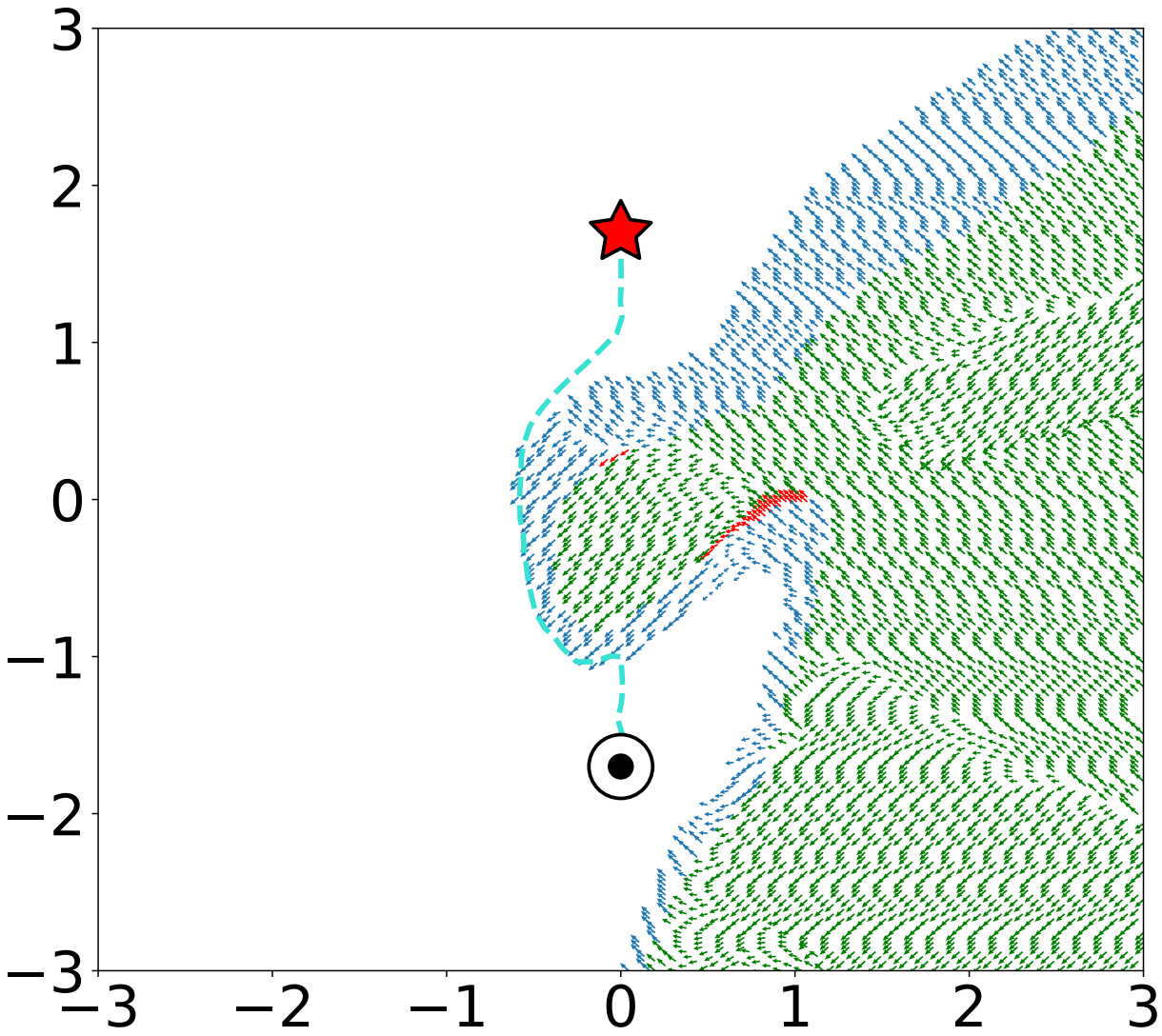


Figure 4.5: We show that the forward-invariance property holds along the trajectory. The direction of each arrow indicates the most admissible action to take on each state x: $\arg\max_{a\in\mathcal{A}} \phi(x,a)$. The green arrows represent the states in the inadmissible region $\{x \mid \forall a \in \mathcal{A}, \phi(x,a) < 0\}$. Both the blue arrows and the red arrows represent the states on the boundary: $\{x \mid \exists a_1, a_2 \in \mathcal{A}, \phi(x,a_1) \geq 0, \phi(x,a_2) < 0\}$. The blue arrows are those boundary states that hold the forward-invariance property. The red arrows are those boundary states that enter the inadmissible region after executing the most admissible action. Note: Since we only train on transitions from collected trajectories, CAM can be asymmetric and violate the forward-invariance property at some unseen states.

Forward-Invariance Property

To further investigate whether the learned CAM holds the forward-invariance property, we illustrate the boundary of the CAM $\{x \mid \exists a_1, a_2 \in \mathcal{A}, \phi(x,a_1) \geq 0, \phi(x,a_2) < 0\}$ in Figure 4.5.

We show that the trajectory never leaves the admissible region of CAM, i.e., the trajectory holds the forward-invariance property. Once the trajectory meets the boundary, it starts to follow the most preferred action among admissible actions, instead of the most preferred action among all sampled actions. Meanwhile, we find that around 0.9% states in the boundary states violate the forward-invariance property, i.e., they enter the inadmissible region after executing the most admissible action.

We emphasize that since we only train on transitions from collected trajectories, CAM can be asymmetric and violate the forward-invariance property at some unseen states. We design the CAM to maintain the forward invariance and focus on states that could exist along possible trajectories. If the agent does not meet some states at any time step during inference, then we do not require these states to obey the forward invariance. Empirically, we have shown that as long as the training tasks and inference tasks are under the same distribution, the forward invariance often holds for inference trajectories once the CAM learns from enough training data.

4.4.2 Additional Single Agent Environment: Dynamic Dubins

In this section, we provide an additional single-agent experiment as a proof of concept. This example has more complex dynamics than the one in the previous section. We provide more details on the experiment setup as follows:

Dynamics. The agent here is one dynamic Dubins' car and controls a 2D action. The state $x = [p_x, p_y, v, \theta], p_x, p_y \in \mathbb{R}, v \in [0,1], \theta \in [0, 2\pi)$, representing the position, velocity, and heading angle in the 2D space. The 2D action $a \in \mathcal{A} = [-1,1]^2$ is composed of the acceleration rate q and the angular velocity $\dot{\theta}$, and the dynamics to compute the next state x' is

$$x' = f(x,a) = x + \begin{bmatrix} v \cdot \sin(\theta + \dot{\theta} \cdot dt) \\ v \cdot \cos(\theta + \dot{\theta} \cdot dt) \\ q \cdot dt \\ \dot{\theta} \cdot dt \end{bmatrix} \tag{4.4}$$

We choose $dt = 0.05$ in our experiment. In this example, we use an MLP to represent CAM, since the state here is solely a vector.

Note that the maximum acceleration rate is saturated here. At each time step, the agent can change the velocity by increasing or decreasing 0.05 at most. As a result, the agent needs to catch failure early on.

Task Distribution. The goal state $(p_x = 0, p_y = 1.7)$, and the obstacles are all fixed during training and test. The starting state of the agent is randomized within the ditribution of $p_x \in [-0.25, 0.25], p_y = -1.7, v = 0, \theta = \frac{\pi}{2}$.

Collision and Goal-reaching Criteria. We define the agent as a circle with a radius of 0.15. The goals are circles with a radius of 0.15. The agent is in a collision if it intersects with the obstacles. If the agent intersects with the 2D goal, then the goal is reached.

Preference Function ω. The preference function calculates the negative of distance to the goal based on the next state: $\omega(x, \mathcal{T}, a) = -||(p'_x, p'_y) - \mathcal{T}||_2^2, p'_x, p'_y \in f(x,a)$.

Forward-Invariance Property

We sample around 40 trajectories starting from the initial states, and illustrate the collected transitions in Figure 4.6. The direction of each arrow indicates the action that CAM agent takes on each state x. The blue arrows represent the admissible region $\{x \mid \forall a \in \mathcal{A}, \phi(x,a) \geq 0\}$. Both the red and the green arrows represent the states on the boundary: $\{x \mid \exists a_1, a_2 \in \mathcal{A}, \phi(x,a_1) \geq 0, \phi(x,a_2) < 0\}$. Only the red arrows represent those boundary transitions that violate the forward-

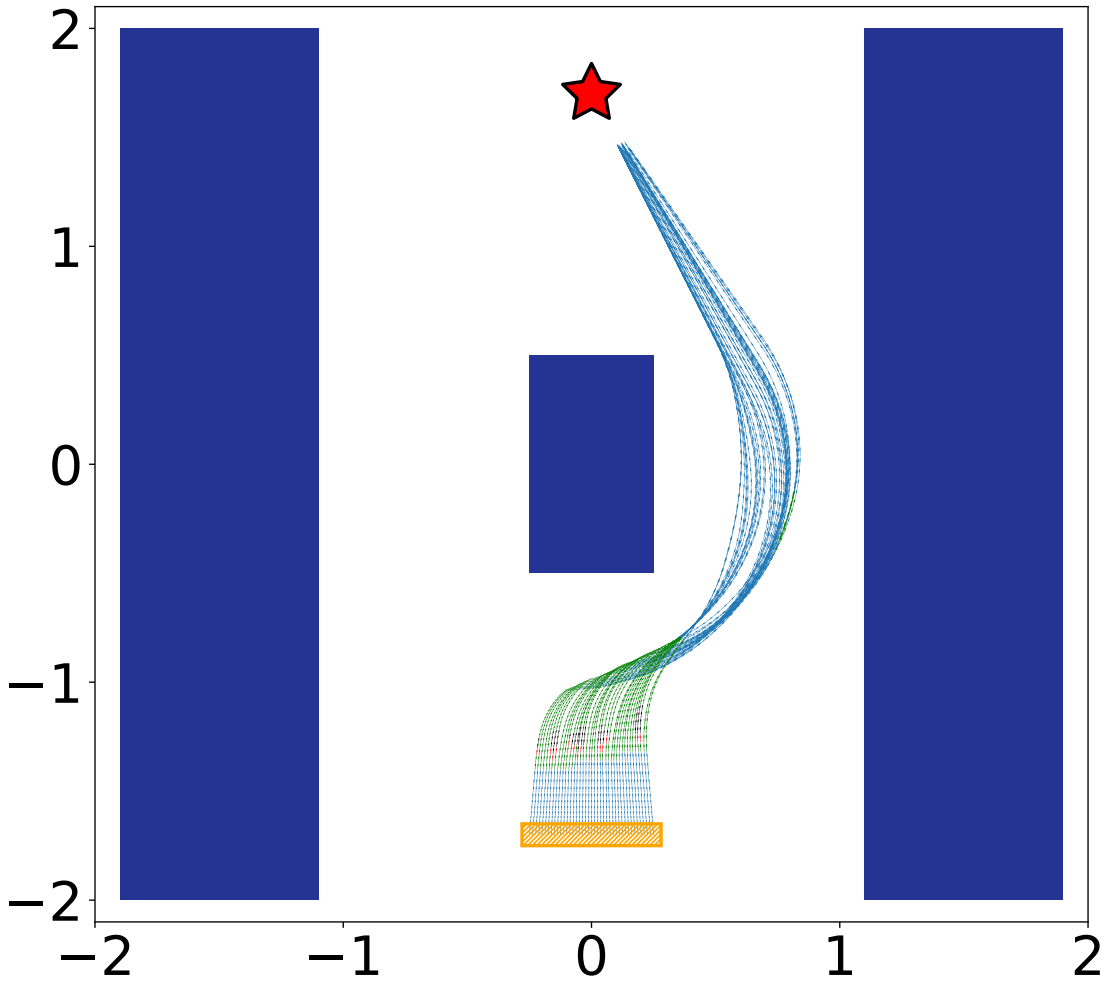


Figure 4.6: An additional single-agent example for the proposed CAM dealing with more complex dynamics, where the agent needs to catch failure early on. The red star represents the goal state, and three blue boxes represent the obstacles. The orange region denotes the set of initial states. We illustrate state-action pairs (x,a) on sampled trajectories starting from initial states. All trajectories reach the goal successfully with no collisions. The direction of each arrow indicates the action that CAM agent takes on each state x. The blue arrows represent the admissible region $\{x \mid \forall a \in \mathcal{A}, \phi(x,a) \geq 0\}$. Both the red and the green arrows represent the states on the boundary: $\{x \mid \exists a_1, a_2 \in \mathcal{A}, \phi(x,a_1) \geq 0, \phi(x,a_2) < 0\}$. Only the red arrows represent those boundary transitions that violate the forward-invariance property. The black arrows represent the states in the inadmissible region $\{x \mid \forall a \in \mathcal{A}, \phi(x,a) < 0\}$.

invariance property. The black arrows represent the states in the inadmissible region $\{x \mid \forall a \in \mathcal{A}, \phi(x,a) < 0\}$.

We show that around 98.6% of transitions along the trajectories are admissible and forward invariant. There are around 27.7% of states on the boundary. Around 0.4% of states violate the forward-invariance property, i.e., the CAM agent crosses the boundary and enters the inadmissible region after executing the action. There are 1% of transitions that are inadmissible, but all of them return back to the admissible regions after executing sequences of most admissible actions.

4.4.3 Details for Multi-Agent Environment

Before proceeding to the main experiment, we provide the details of the multi-agent environments we used in our experiments. We design four different environments, including UR5, Car, Dynamic Dubins, and Drone. Each environment has its own dynamics, graph representation, training distribution, collision and goal-reaching criteria, preference function ω, and hyperparameters. The details are described as follows:

Details for the UR5 Environment

Dynamics. Each UR5 agent controls a 5D robot arm. The state $x = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5] \in [-2\pi, 2\pi]^5$, representing the angles on the joints of base to shoulder, shoulder to lift, elbow, wrist 1, and wrist 2. The action is the angular velocity $a = [\dot{\theta_1}, \dot{\theta_2}, \dot{\theta_3}, \dot{\theta_4}, \dot{\theta_5}] \in [-1, 1]^5$, and the dynamics to compute the next state x' are

$$x' = f(x, a) = x + a \cdot dt \tag{4.5}$$

We choose $dt = 0.1$ in our experiment.

Graph Representation. Each node $v \in V$ is simply a zero scalar. For each agent. Each edge $e : (v_i, v_j) \in E$ is a 18D vector, which concatenates 4 vectors. The first 4D vector is a one-hot vector representing the arm ID of v_i. The second 5D vector represents the state of the arm v_i. The third and fourth vectors follow the same representation as to the first and the second vectors, but

those of arm v_j replace the values. The graph is complete for this environment, i.e., each arm can affect any other arm in the environment.

Training Distribution. The training distribution is that only two arms should be in the graph.

Collision and Goal-reaching Criteria. We define collision as the contact or penetration between one arm and another arm or static obstacle. If the state of the arm is within 0.3 distance of the goal, then the goal is reached.

Preference Function ω. The preference function calculates the negative of distance to the goal based on the next state: $\omega(G, \mathcal{T}, a) = -||f(x_G, a) - \mathcal{T}||_2^2$.

Hyperparameters. For each time step, we sample $N = 2000$ candidate actions for every agent. The initial learning rate of the ADAM optimizer is 3e-4, and we decrease the learning rate with a factor of 0.5 when the performance in the validation environment meets a plateau until we reach a minimum learning rate of 1e-4. We set $\gamma_1, \gamma_2, \gamma_3$ as 0, 1e-1, 1e-2. We use a batch size of 256 and update the network every 20 trajectories.

Details for the Car Environment

Dynamics. Each Car agent follows the Dubins' Car dynamics [42]. The state $x = [p_x, p_y, \theta], p_x, p_y \in \mathbb{R}, \theta \in [0, 2\pi)$, representing the positions on 2D plane and the heading angle. We apply a constant $v = 0.05$ as the velocity for each agent. The action is the angular velocity $a = [\dot{\theta}] \in [-\frac{2}{3}\pi, \frac{2}{3}\pi]$, and the dynamics to compute the next state x' are

$$x' = f(x, a) = x + \begin{bmatrix} v \cdot \sin(\theta + \dot{\theta} \cdot dt) \\ v \cdot \cos(\theta + \dot{\theta} \cdot dt) \\ \dot{\theta} \cdot dt \end{bmatrix} \tag{4.6}$$

We choose $dt = 1$ in our experiment.

Graph Representation. Each node $v \in V$ is a one-hot vector indicating whether the current node is an agent or a static obstacle. Given each edge $e : (v_i, v_j) \in E$, the feature vector of the edge is

$[I, \sin(\theta_i), \cos(\theta_i), \sin(\theta_j), \cos(\theta_j), p_{x,i} - p_{x,j}, p_{y,i} - p_{y,j}]$, where I is a one-hot vector, representing whether the current edge is obstacle-to-agent or agent-to-agent. For the obstacle-to-agent edges, $\sin(\theta_i), \cos(\theta_i)$ are padded as 0.

We define each node's neighbors, including agents and obstacles, as those nodes having a distance within 1.5 to the current node. We only connect edges from these neighbor nodes to each node.

Training Distribution. The training distribution is that only 0-2 neighbor agents and 0-9 neighbor obstacles should be in the graph.

Collision and Goal-reaching Criteria. We define the agent and the obstacle as circles with a radius of 0.15. The goals are circles with a radius of 0.3. The agent is in a collision if it is within a distance of 0.3 to other agents or obstacles. If the agent's position is within a distance of 0.45 to the 2D goal, then the goal is reached.

Preference Function ω. The preference function calculates the negative of distance to the goal based on the next state: $\omega(G, \mathcal{T}, a) = -||(p'_x, p'_y) - \mathcal{T}||_2^2, p'_x, p'_y \in f(x_G, a)$.

Hyperparameters. For each time step, we sample $N = 2000$ candidate actions for every agent. The initial learning rate of the ADAM optimizer is 1e-3, and we decrease the learning rate with a factor of 0.5 when the performance in the validation environment meets a plateau until we reach a minimum learning rate of 1e-5. We set $\gamma_1, \gamma_2, \gamma_3$ as 0, 2e-2, 1e-2. We use a batch size of 256 and update the network every 10 trajectories.

Details for the Dynamic Dubins Environment

Dynamics. The agent follows dynamic Dubins' car and controls a 2D action. The state $x = [p_x, p_y, v, \theta], p_x, p_y \in \mathbb{R}, v \in [0, 1], \theta \in [0, 2\pi)$, representing the position, velocity, and heading angle in the 2D space. The 2D action $a \in \mathcal{A} = [-1, 1]^2$ is composed of the acceleration rate q and the angular velocity $\dot{\theta}$, and the dynamics to compute the next state x' is

$$x' = f(x,a) = x + \begin{bmatrix} v \cdot \sin\left(\theta + \dot{\theta} \cdot dt\right) \\ v \cdot \cos\left(\theta + \dot{\theta} \cdot dt\right) \\ q \cdot dt \\ \dot{\theta} \cdot dt \end{bmatrix} \tag{4.7}$$

We choose $dt = 0.05$ in our experiment.

Graph Representation. Each node is a one-hot vector indicating whether the current node is an agent or a static obstacle. Given each edge $e : (i, j) \in E$, the feature vector of the edge is $[I, v_i, v_j, \sin(\theta_i), \cos(\theta_i), \sin(\theta_j), \cos(\theta_j), p_{x,i} - p_{x,j}, p_{y,i} - p_{y,j}]$, where I is a one-hot vector, representing whether the current edge is obstacle-to-agent or agent-to-agent. For the obstacle-to-agent edges, $v_i, \sin(\theta_i), \cos(\theta_i)$ are padded as 0.

We define each node's neighbors, including agents and obstacles, as those nodes having a distance within 1.5 to the current node. We only connect edges from these neighbor nodes to each node.

Training Distribution. The training distribution is that only 0-2 neighbor agents and 0-9 neighbor obstacles should be in the graph.

Collision and Goal-reaching Criteria. We define the agent and the obstacle as circles with a radius of 0.15. The goals are circles with a radius of 0.3. The agent is in a collision if it is within a distance of 0.3 to other agents or obstacles. If the agent's position is within a distance of 0.45 to the 2D goal, then the goal is reached.

Preference Function ω. The preference function calculates the negative of distance to the goal based on the next state: $\omega(G, \mathcal{T}, a) = -||(p'_x, p'_y) - \mathcal{T}||_2^2, p'_x, p'_y \in f(x_G, a)$.

Hyperparameters. For each time step, we sample $N = 2000$ candidate actions for every agent. The initial learning rate of the ADAM optimizer is 1e-3, and we decrease the learning rate with a factor of 0.5 when the performance in the validation environment meets a plateau until we reach a

minimum learning rate of 1e-5. We set $\gamma_1, \gamma_2, \gamma_3$ as 0, 2e-2, 1e-2. We use a batch size of 256 and update the network every 10 trajectories.

Details for the Drone Environment

Dynamics. Each Drone agent follows a 3D quadrotor model adopted from [34] and controls a 4D action. The state $x = [p_x, p_y, p_z, v_x, v_y, v_z, \alpha, \beta, \gamma], p_x, p_y, p_z \in \mathbb{R}, v_x, v_y, v_z \in [-1, 1], \alpha, \beta, \gamma \in [-\pi/2, \pi/2]$, representing the position and velocity in the 3D space, and the angles in pitch, roll, and yaw. The action is the instant acceleration q and the angular velocities, i.e., $a = [q, \dot{\alpha}, \dot{\beta}, \dot{\gamma}] \in [-1, 1]^4$, and the dynamics to compute the next state x' is

$$
\begin{aligned}
f(x, t+dt) &= f(x,t) + \begin{bmatrix} v_x \\ v_y \\ v_z \\ -\sin(\beta) \cdot q \\ \cos(\beta)\sin(\alpha) \cdot q \\ \cos(\beta)\cos(\alpha) \cdot q - g \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} \cdot dt \\
f(x, 0) &= x \\
x' &= f(x, k \cdot dt)
\end{aligned}
\tag{4.8}
$$

g is the gravity acceleration of the Earth as 9.8. We choose $dt = 0.01$ and $k = 10$ in our experiment.

Graph Representation. Each node $v \in V$ is a one-hot vector indicating whether the current node is an agent or a static obstacle. Given each edge $e : (v_i, v_j) \in$

E, the feature vector of the edge is $[I, v_{x,i}, v_{y,i}, v_{z,i}, \sin(\alpha_i), \cos(\alpha_i), \sin(\beta_i), \cos(\beta_i), v_{x,j}, v_{y,j}, v_{z,j}, \sin(\alpha_j), \cos(\alpha_j), \sin(\beta_j), \cos(\beta_j), p_{x,i} - p_{x,j}, p_{y,i} - p_{y,j}, p_{z,i} - p_{z,j}]$, where I is a one-hot vector, representing whether the current edge is obstacle-to-agent or agent-to-agent. For the obstacle-to-agent edges, $v_{x,i}, v_{y,i}, v_{z,i}, \sin(\alpha_i), \cos(\alpha_i), \sin(\beta_i), \cos(\beta_i), p_{z,i} - p_{z,j}$ are padded as 0.

We define each node's neighbors, including agents and obstacles, as those nodes having a distance within 1.5 to the current node. We only connect edges from these neighbor nodes to each node.

Training Distribution. The training distribution is that only 0-2 neighbor agents and 0-9 neighbor obstacles should be in the graph.

Collision and Goal-reaching Criteria. We define the agent as spheres with a radius of 0.15. The obstacles are vertical cylinders with a radius of 0.15 and infinite height. The goals are spheres with a radius of 0.3. The agent is in a collision if it has a distance within 0.3 to other agents or obstacles in the 3D space. An agent reaches its goal if its 3D position has a distance of 0.45 to its goal.

Preference Function ω. The preference function calculates the negative of an error between the action a and a linear–quadratic regulator (LQR) controller π. The LQR controller π takes the goal and the current state, and gives an action which only considers the goal-reaching: $\omega(G, \mathcal{T}, a) = -||\pi(x_G, \mathcal{T}) - a||_2^2$.

Hyperparameters. For each time step, we sample $N = 2000$ candidate actions for every agent. The initial learning rate of the ADAM optimizer is 1e-3, and we decrease the learning rate with a factor of 0.5 when the performance in the validation environment meets a plateau until we reach a minimum learning rate of 1e-5. We set $\gamma_1, \gamma_2, \gamma_3$ as 0, 1e-1, 1e-2. We use a batch size of 256 and update the network every 20 trajectories.

4.4.4 Multi-agent Navigation Experiments

Experiment Setup. We evaluate our methods under 4 types of environments: UR5, Car, Dynamic Dubins, and Drone. We briefly describe these 4 environments as follows: **(i)** The UR5 environment is implemented with PyBullet [31] and contains 4 decentralized 5D robot arms with cubic obstacles. We randomly select solely 2 robot arms from these four arms during training. We fix the configurations of the static obstacles and the goal states of the arms. In order to make the test task solvable, the initial states are randomized during training and fixed during testing. **(ii)** For the Car environment, each agent follows the 2D Dubins' car model [42], and controls a 1D action. The static obstacles are circles with a fixed radius. **(iii)** The Dynamic Dubins agent controls a 2D action and follows a modified Dubins' Car dynamic which enables the control of acceleration. **(iv)** For the Drone environment, each agent follows a 3D quadrotor model adopted from [34] and controls a 4D action. Each obstacle is a cylinder with infinite height. For both Car and Drone environments, we only train on environments with 3 agents, randomized initial states and goal states, and a set of random obstacles. The map size is fixed as 3x3 during training while varying during testing.

To evaluate our CAM approach thoroughly, we design diverse scenarios for all four environments. For the UR5 environment, we make the number of agents vary from 2 to 4. For scenarios with 2 or 3 arms, the arms are selected randomly from the four decentralized arms. For the other environments, we fix the map size as 32x32 and vary the ratio of agents over obstacles from 1:1023 to 512:512. For each setting, we average the performance over 100 randomly generated test cases.

Baselines. The baseline approaches we compare with include DDPG [103] and MACBF [132]. DDPG is a standard RL approach to learning safe behavior through maximizing rewards. MACBF is a state-of-the-art approach for multi-agent safe control problems, which learns a policy jointly with neural safe certificates. We reimplement all the baselines with GNNs for a fair comparison. Note that our GNN implementation for DDPG is a multi-agent algorithm, since each agent is

aware of the other agents by taking the egocentric graph as the input.

Evaluation Metrics. We evaluate our methods based on two metrics: the safe rate and the average reward. The safe rate counts the number of collisions for each agent at every time step and averages over all the agents through the whole trial [132]. At each time step, the reward is -1 if the agent collides with another agent or the static obstacle. A +10 reward will be given at the end of the trajectory, if the agent reaches its goal at any time step along the trajectory. A small negative constant of -0.1 is added to the reward to encourage shorter paths for goal-reaching. The reward is accumulated throughout the trial and averaged over all the agents.

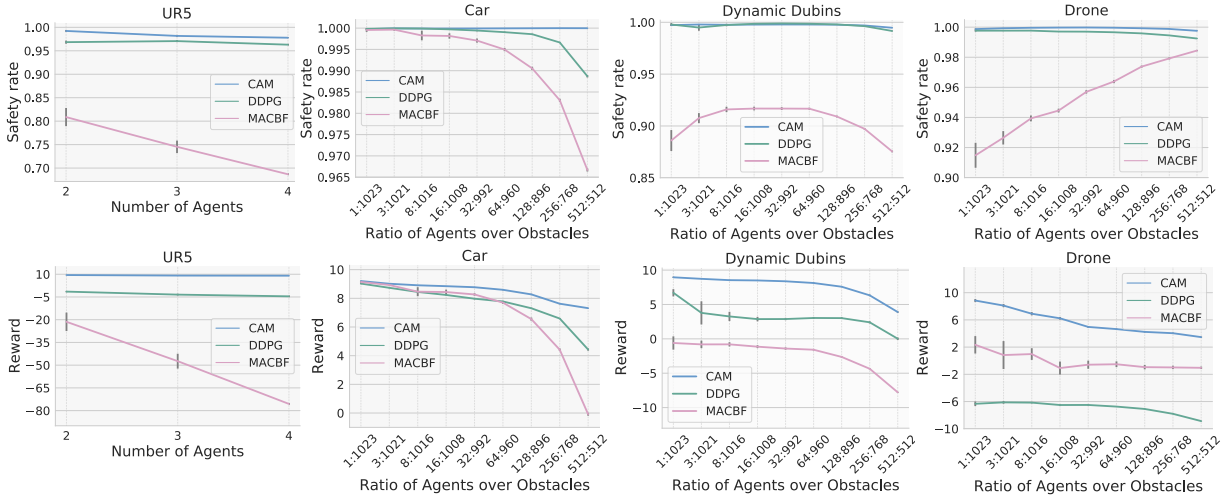


Figure 4.7: We show the performances of CAM and prior methods in the UR5, Car, Dynamic Dubins, and Drone environment, where each method is trained with 3 agents during training but tested up to 512 agents. CAM substantially outperforms prior approaches in all the environments.

Overall Performance. In Figure 4.7, we demonstrate the overall performances for the UR5, Car, Dynamic Dubins, and Drone environments. Our method shows the generalization across different environments and densities of agents and obstacles. Though trained with only 3 agents, the safety rate of CAM remains nearly 100% for all the environments, up to 512 agents. Even if the density of agents and obstacles deviates from the training environment, our CAM approach can avoid the distribution shift, using decomposition on large graphs and the compositionality of CAMs.

On the other side, though our implementation for MACBF and DDPG can take an arbitrary number of agents using GNNs, the performance degrades significantly when the distribution of

input graphs shifts in test environments. We also inspect that the averaged reward for all the methods decreases in most cases when the number of agents increases. It is due to more agents failing to reach the goal when the timeout happens since it takes more time for agents to avoid each other. We encourage the reader to view the video demonstrations in the supplementary material for all 4 environments.

4.4.5 Notes on the Multi-agent Navigation Experiments

Reproducing the Baselines. The inputs to the networks of DDPG and MACBF are the same as those to the CAM. For DDPG, we use the GNN architectures as the actor and Q-critic network. The GNN actor takes an egocentric graph for each agent and outputs the action. The GNN critic takes the graph and action and outputs the Q value. For MACBF, since it requires differentiable dynamics, we implement an additional dynamic function using PyTorch to guarantee the differentiability. When computing the derivative of the CBF value over time, we assume the graph structure remains the same, i.e., no adding or removing edges. We also find that replacing the temporary dataset of MACBF with the replay buffer, typically used for the RL methods, improves the stability of MACBF.

Notes on why MA-DDPG is not deployed here [107]: Our GNN implementation deviates from the MA-DDPG algorithm because of scalability concerns. In MA-DDPG, the actor network for each agent is independent and does not share the weight. As a result, MA-DDPG requires the same number of agents for the training and inference tasks. However, in our experiment, since we focus on the generalization for scalability, the number of agents during inference varies, while our training tasks are fixed to have three agents. Thus, it is impossible to deploy the MA-DDPG algorithm.

Running the Experiments. All the methods are trained on CPU as Intel Core i7-11700F. The GPU is NVIDIA RTX 3080. We train each method until we reach convergence, which typically requires fewer than two days.

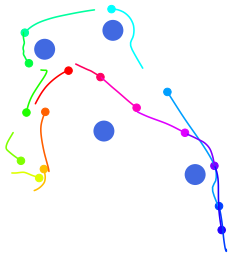
Accelerating the Computation. Though the GNN can be deployed on decentralized devices at inference time, we conduct our experiment on one single desktop, which requires computation optimization. Assume at a time step of the inference time that there are 512 agents in the environment. We need to test 2000 actions for each agent, and each agent samples at least 100 subgraphs. We can infer that there are at least $100 \times 2000 \times 512 \approx 1.02 \times 10^8$ forward operations for the GNN to compute. We need to accelerate such computation; otherwise, the computational cost will be intractable.

We conduct three improvements for the acceleration. The first improvement is to merge the computation of the hidden state for all state-action pairs which share the same input graph into one forward passing in the GNN layer, as mentioned in Section 4.3.2. The second improvement is to parallelize the computation of the CAM value by stacking the pairs of hidden states and actions as a matrix and then feeding it to the GPU. Such stacking enables the computation with batches of state-action pairs. The third improvement is to choose which agents to compute for each batch adaptively. By inspecting the CAM values of the computed batches, if we have already found admissible actions for one specific agent, then the rest of the state-action pairs for this agent do not need to be computed, and the agent chooses the action among those admissible actions that have been computed. By filtering out the determined agents, we save the unnecessary computation of the state-action pairs of these agents.

4.4.6 Zero-Shot Transfer to Chasing Game

In this section, we study the generalization of our CAM model for zero-shot transfer to other multi-agent tasks, e.g., the chasing game. In the chasing game, each agent chases another agent to pursue. The target agent is assigned randomly at the beginning of every episode. The agent's goal is to maintain the distance to the assigned agent, with no collision with agents and obstacles. We fix the number of agents to 64 in the experiment. This new task is substantially different from the task for training, where the assigned goal is fixed. We define the safety rates

Table 4.1: Left: A snapshot of a trajectory in the chasing game with 16 Drone agents from a bird's-eye view. The agents are the colorful points, and the obstacles are the blue circles of a larger size. Each agent aims to chase another agent without collision. Right: Performances of CAMs with and without graph decomposition on 64 agents, which shows the effectiveness of the graph decomposition.



Method	**CAM**	**CAM w/o decomposition**
Safety Rate (Car)	0.99±0.00	0.90±0.03
Reward (Car)	3.10±1.20	-19.98±7.08
Safety Rate (Drone)	0.989± 0.005	0.988±0.008
Reward (Drone)	6.84±1.12	6.54±2.15

of the chasing game in the same way as those of the navigation task. The reward is defined in another way, to reflect on the situation that one Drone agent could drift away from its target agent though it executes the most preferred action. We want to avoid penalizing such behavior because the most preferred action should not be discouraged if it is also safe at the same time. We define reward as follows: $R = \sum_{t=1}^{T} \text{clip}(d_{t-1} - d_t, 0, 2)$, where d_{t-1} and d_t indicate the distances between the agent and the target agent in 2D or 3D space at $t-1$-th and t-th step. Such a reward keeps track of the agent's improvement at every step while not penalizing on no progress. The preference for the agent follows the same way we defined in the navigation tasks of the Car and Drone. We update the goal for each agent to be the position of the target agent at every step.

In Table 4.1, we show that our CAM method can adapt to the new task effectively due to its safety and goal-reaching decoupling property, preserving a relatively high safety rate and reward. In addition, we compare our method to the CAM without the graph decomposition. We observe that the graph decomposition works effectively in both environments. The performance of the Car environment benefits more from the graph decomposition. This advancement is because the density of Car agents in 2D is higher than that of the Drone agent in 3D. The higher density makes the test task more unlike the training task, which explains why the graph decomposition is more beneficial for Car.

4.5 Ablation Study: Varying Obstacle Density

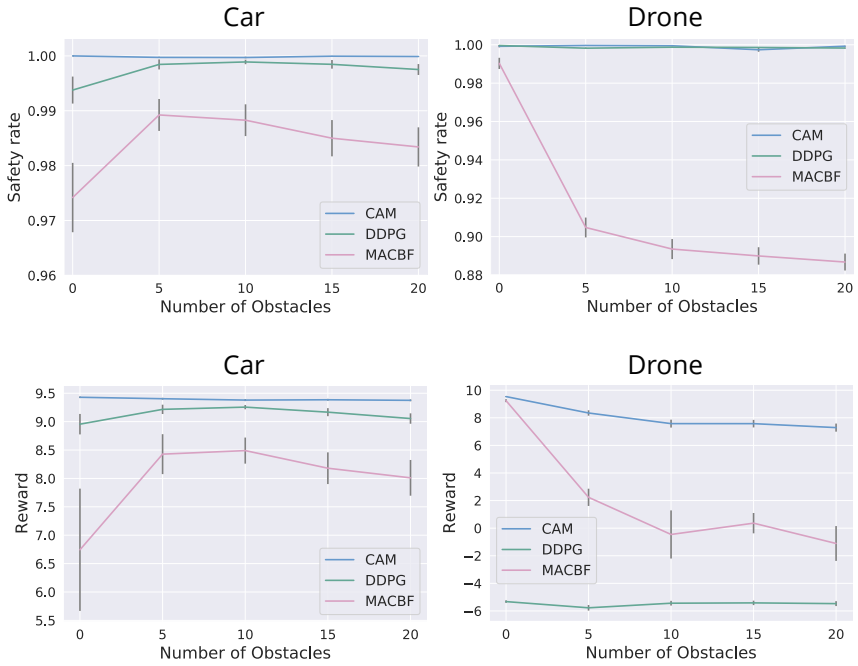


Figure 4.8: The performance of all methods on Car and Drone with regard to varying obstacle numbers.

In the main paper, we test the scenarios where the densities of both the agents and the obstacles vary. In this section, we focus on how the obstacle density affects the performance of each method.

We fixed the map size to be 4x4 and increased the obstacle number from 0 to 20. We average the results over 100 randomly generated environments. We do not include the UR5 environment since the obstacle number for UR5 is fixed to be 4. As shown, the CAM significantly outperforms other methods and maintains the safety rate near 100%. On the other hand, MACBF and DDPG show a lack of robustness when the obstacle density increases. We also observe a remarkable degradation of the performance of MACBF for the Drone environment. When there are 0 obstacles, the safety rate of MACBF is close to 99%, but then the safety rate jumps to around 90% when there are 5 obstacles. Such a result shows that MACBF works better to avoid inter-agent collisions and cannot deal well with obstacle-agent collision avoidance. This result

explains why the performance of MACBF improves while the density of obstacles decreases in the Drone environment in Figure 4.7.

4.6 Ablation Study: Inference Time

Figure 5.5 illustrates the running time during inference. We average the running time over all the agents and all the time steps in the test cases. With the three acceleration improvements mentioned in Section 4.4.5, we can now reduce the running time to around 4.3 milliseconds at most. The computation time can further be improved when deployed in real-world experiments. In that case, each agent can compute the admissible set independently, since our GNN architecture is fully decentralized. It only requires observations of nearby agents and obstacles and does not need inter-agent communication.

We find that in Figure 5.5, the trends of the running time for Car and Drone are similar - the running time decreases and then increases as the density of agents grows. The running time first decreases, since our method utilizes the GPU and computes the admissible actions of all the agents in parallel. However, when the number of agents grows to a certain threshold between 32 and 128, the memory of the GPU will reach its limitation and cannot take more agents beyond this threshold. If the number of agents is higher than this threshold, we need multiple GPU forward passes to cover all the agents. As a result, the running time increases afterward. More GPU memory should make the computation even faster in such a centralized computation scenario.

4.7 Limitations and Failure Modes

While the framework is generalizable in terms of scalability, it has limitations. Since the proposed GNN architecture is entirely decentralized, it disables the agents from communicating and coordinating with each other in the swarm. As a result, it could be challenging for agents to

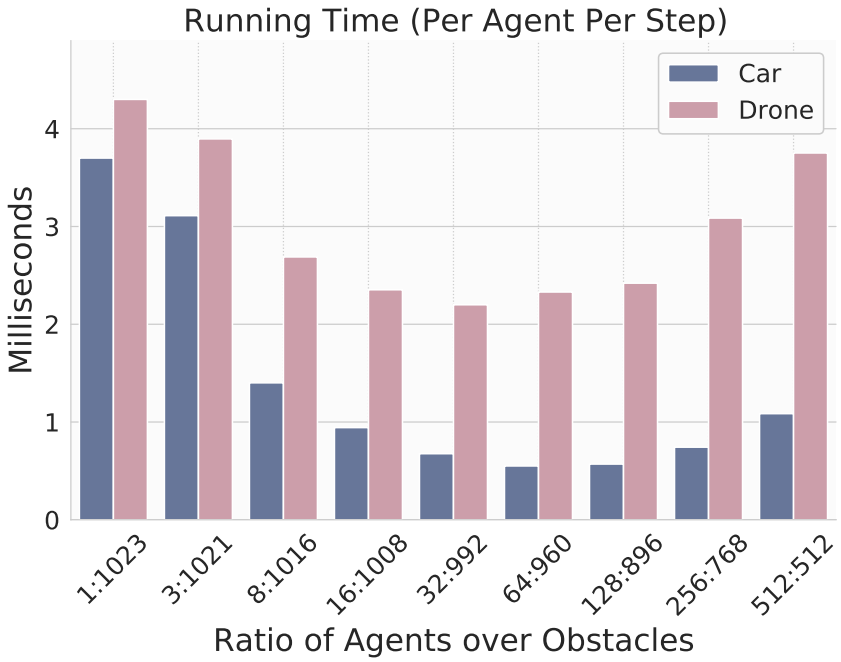


Figure 4.9: The average running time of our method on Car and Drone during inference for each agent at each time step (in milliseconds). The running time is averaged over the 100 test cases.

identify and avoid critical situations requiring interactions among agents, e.g., dead-locks and states with very few feasible actions. As shown in Figure 4.7, there is a slight drop in the safety rate of the Drone environment when the number of agents increases to 512. To understand such a failure mode, we inspect these trajectories and find that the corresponding CAM values are all negative at several consecutive preceding states right before the collision. Such behavior indicates a very low probability of finding feasible actions in these states, which validates the necessity for inter-agent communication or global coordination when the density of agents is very high in local regions. We believe that a preference function considering the global coordination can be helpful for this issue.

4.8 Conclusion

We proposed new learning-based control methods for multi-agent navigation problems by shifting the focus from training optimal control policies to training CAMs that implicitly represent a set of feasible actions. We avoid the reward engineering by decoupling the multi-agent navigation tasks using a simple goal-reaching preference function and a learnable CAM for collision avoidance. Our methods use GNNs to represent the CAM which takes egocentric graphs as inputs and proposes the relabelling with backtracing to learn from rich information even given sparse reward. In online inference, we propose the graph decomposition to deal with the covariate shift, which achieves a notably high success rate and low collision rate. While unconventional compared to the types of models routinely used in multi-agent reinforcement learning, we demonstrate that CAM has several useful properties and is particularly effective in multi-agent settings that require compositionality and generalization for scaling at inference time. In future work, we will study how to incorporate multi-agent communication and test its effectiveness in real-world scenarios.

4.9 Acknowledgments

Chapter 4, in full, is a reprint of Chenning Yu, Hongzhan Yu, Sicun Gao, "Learning Control Admissibility Models with Graph Neural Networks for Multi-Agent Navigation", CoRL, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Improving Compositional Generation with Diffusion Models Using Lift Scores

We introduce a novel resampling criterion using lift scores, for improving compositional generation in diffusion models. By leveraging the lift scores, we evaluate whether generated samples align with each single condition and then compose the results to determine whether the composed prompt is satisfied. Our key insight is that lift scores can be efficiently approximated using only the original diffusion model, requiring no additional training or external modules. We develop an optimized variant that achieves minimal computational overhead during inference while maintaining effectiveness. Through extensive experiments, we demonstrate that lift scores significantly improve compositional generation across 2D synthetic data, CLEVR position tasks, and text-to-image synthesis.

5.1 Introduction

Despite the success of diffusion models in generating high-quality images [153, 64, 121, 155], they sometimes fail to produce coherent and consistent results when the condition

is complex or involves multiple objects [68, 48, 24]. This limitation is particularly evident in tasks that generate the images with specific combinations of attributes or objects, where the generated samples may only partially satisfy the conditions or exhibit inconsistencies across different attributes.

To mitigate this issue, several methods have been proposed to improve the compositional generation with diffusion models. These methods can be described as 2 main categories: (1) **training-based** methods, which train or finetune the diffusion model to improve its ability to generate samples that satisfy complex prompts [68, 7], and (2) **training-free** methods, which either refine the sampling process, or apply rejection sampling to refine the generated samples without modifying the underlying model [48, 24]. These 2 categories of methods are not necessarily mutually exclusive, and they can be combined to achieve better performance.

We introduce *CompLift*, a training-free rejection criterion for improving compositional generation in diffusion models. The concept of lift score [19] builds on a simple observation: if a generated sample truly satisfies a condition, the model should perform better at denoising it when given that condition compared to unconditional denoising. This insight leads to an efficient criterion for accepting or rejecting samples that requires minimal computational overhead. Similar to Diffusion Classifier [98], we show that lift scores can be efficiently approximated using only the original diffusion model's predictions, requiring no additional training or external modules. By decomposing the satisfaction of a complex prompt into the satisfaction of multiple simpler sub-conditions using lift scores, our approach can boost the model's alignment with complex prompts without changing the sampling process. Through extensive experiments on 2D synthetic data, CLEVR position tasks, and text-to-image generation, we demonstrate that *CompLift* significantly improves compositional generation while maintaining computational efficiency.

Contributions. Our contributions are threefold: ❶ We introduce *CompLift*, as a novel resampling criterion using lift scores for compositional generation, requiring minimal computational overhead and no additional training (Section 5.3). ❷ We explore the design space of *CompLift*, including

noise sampling strategies, timestep selection, and caching techniques to optimize computational efficiency (Section 5.4). ❸ We scale our approach to text-to-image generation, demonstrating its effectiveness in improving compositional generation in this challenging domain (Section **??**).

5.2 Related Work

Recent studies have revealed significant limitations in diffusion models [36] for handling compositional generation, particularly as missing objects, incorrect spatial relationships, and attribute inconsistencies [68, 48, 24].

Current approaches to address these challenges fall into several categories. One line of work leverages the *cross-attention* mechanisms, modifying attention values during synthesis to emphasize overlooked tokens [48, 24, 175, 174]. Another approach introduces *layout control* for precise spatial arrangement of objects [49, 27]. While effective, these methods require specific model architectural knowledge, unlike our model-agnostic approach.

The *diffusion process* itself can also be modified to improve compositional generation. Composable Diffusion [105] guides generation by combining individual condition scores. Discriminator Guidance [86] employs an additional discriminator to reduce the distribution mismatch of the generated samples. Recent methods introduce advanced sampling techniques like Markov Chain Monte-Carlo [41], Restart Sampling [178], rejection sampling [120], and particle filtering [106]. We complement these methods by providing a technique that can be integrated without assumptions about the underlying sampling process.

Most closely related to our work are *resampling strategies* that operate as either selection of the final image [80] or search of the initial noise [131, 111]. However, these methods typically require external models for quality assessment. In contrast, our approach leverages the denoising properties of the diffusion model itself [98, 26, 90], eliminating the need for additional models.

5.3 Using Lift Scores for Rejection Sampling

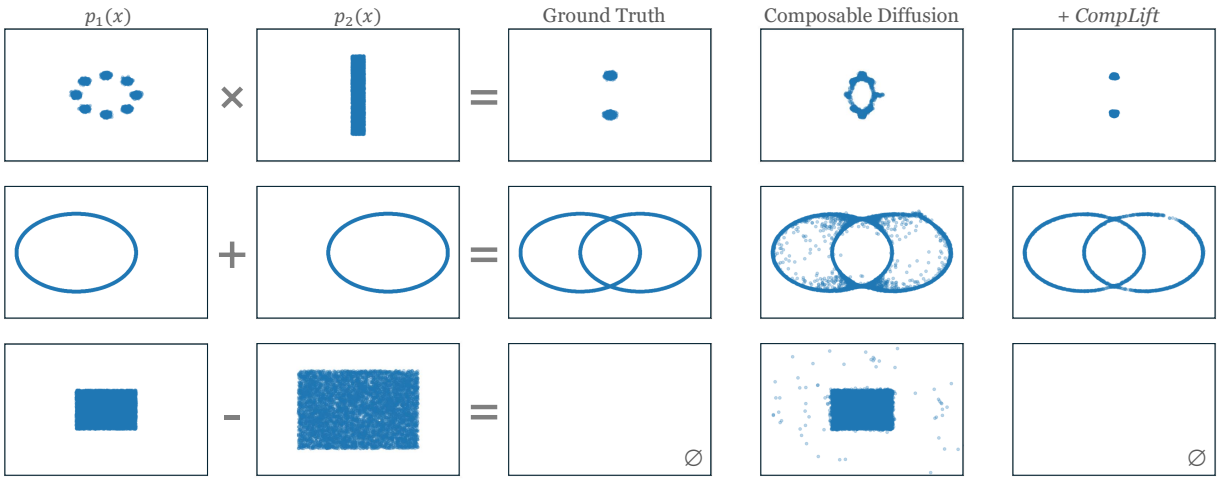


Figure 5.1: **An illustration of product, mixture, and negation compositional models, and the improved sampling performance using *CompLift*.** Left to right: Component distributions, ground truth composed distribution, composable diffusion samples, samples accepted by *CompLift*. Top: product, center: mixture, bottom: negation. ∅ represents the empty set - no samples are generated or accepted.

5.3.1 Diffusion Model Preliminaries

Diffusion model is a class of generative models that generate samples by denoising a noisy input iteratively. Given a clean sample $\boldsymbol{x}_0$, the forward diffusion process $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ sequentially adds Gaussian noise to $\boldsymbol{x}_{t-1}$ at each timestep t, whereas the learned reverse diffusion process $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{c})$ denoises $\boldsymbol{x}_t$ to reconstruct $\boldsymbol{x}_{t-1}$, optionally conditioned on a condition $\boldsymbol{c}$. The conditional probability of $\boldsymbol{x}_0$ can be defined as:

$$p_\theta(\boldsymbol{x}_0|\boldsymbol{c}) = \int_{\boldsymbol{x}_{1:T}} p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{c})\, d\boldsymbol{x}_{1:T},$$

where $p(\boldsymbol{x}_T)$ is typically a standard Gaussian $\mathcal{N}(\mathbf{0},\mathbf{I})$.

The diffusion model is often parameterized as a neural network ε_θ to represent the denoising function. Using the fact that $q(\boldsymbol{x}_t|\boldsymbol{x}_0) := \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$, the model is

Algorithm 8 Rejection with *CompLift*

Input: test sample $\boldsymbol{x}_0$, condition $\{\boldsymbol{c}_i\}_{i=1}^n$, # of trials T, algebra $\mathcal{A}$
Initialize $\texttt{Lift}[\boldsymbol{c}_i] = \text{list}()$ for each $\boldsymbol{c}_i$
for trial $j = 1, \ldots, T$ **do**
  Sample $t \sim [1, 1000]$; $\varepsilon \sim \mathcal{N}(0, I)$
  $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$
  for condition $\boldsymbol{c}_k \in \{\boldsymbol{c}_i\}_{i=1}^n$ **do**
    $\mathit{lift}_j(\boldsymbol{x}_0|\boldsymbol{c}_i) = \|\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \varnothing)\|^2 - \|\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c}_k)\|^2$
    $\texttt{Lift}[\boldsymbol{c}_k].\texttt{append}(\mathit{lift}_j(\boldsymbol{x}_0|\boldsymbol{c}_i))$
$\{\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i)\}_{i=1}^n = \{\texttt{mean}(\texttt{Lift}[\boldsymbol{c}_i])\}_{i=1}^n$
return $\texttt{Compose}(\{\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i)\}_{i=1}^n, \mathcal{A})$ ▷ Table 5.1

trained by maximizing the variational lower bound (VLB) of the log-likelihood of the data, which can be approximated as the Evidence Lower Bound (ELBO), i.e., diffusion loss:

$$\begin{aligned}\log p_\theta(\boldsymbol{x}_0|\boldsymbol{c}) &\geq \mathbb{E}_q\left[\log \frac{p_\theta(\boldsymbol{x}_{0:T}, \boldsymbol{c})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right] \\ &= -\mathbb{E}_{\varepsilon,t}\left[w_t\|\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})\|^2\right] + C.\end{aligned}$$

Following the previous works [64, 98, 26], we set $w_t = 1$ for all t as it achieves good performance in practice.

5.3.2 Approximating Lift Scores

Given a sample $\boldsymbol{x}$, we wish to check whether it aligns with a condition $\boldsymbol{c}$. In other words, we want to see how well the association $\boldsymbol{x}$ has with $\boldsymbol{c}$. This brings us to the concept of *lift*, which is an existing concept in data mining:

$$\mathit{lift}(\boldsymbol{x}|\boldsymbol{c}) := \log \frac{p(\boldsymbol{c}|\boldsymbol{x})}{p(\boldsymbol{c})} = \log \frac{p(\boldsymbol{x}, \boldsymbol{c})}{p(\boldsymbol{x})p(\boldsymbol{c})} = \log \frac{p(\boldsymbol{x}|\boldsymbol{c})}{p(\boldsymbol{x})}. \tag{5.1}$$

We note that the definition of lift scores is slightly different from the traditional data mining definition [19] as it is defined in the logarithmic space. We abuse the notation since it is easier to

implement in the logrithm space, and we call it *lift score* throughout this paper.

One perspective to understand lift scores is to regard p as a perfectly-learned classifier. If $\boldsymbol{x}$ is an arbitrary unknown sample, this classifier tries its best to predict the probability of $\boldsymbol{x}$ belonging to class $\boldsymbol{c}$ as $p(\boldsymbol{c})$. However, once $\boldsymbol{x}$ is given as a known sample, the classifier will utilize the information in $\boldsymbol{x}$ to increase its prediction accuracy, as $p(\boldsymbol{c}|\boldsymbol{x})$. If $\boldsymbol{x}$ has any positive association with $\boldsymbol{c}$, we would expect $p(\boldsymbol{c}|\boldsymbol{x}) > p(\boldsymbol{c})$, which means $\mathit{lift}(\boldsymbol{x}|\boldsymbol{c}) > 0$.

Approximating the lift scores. In preliminaries, we see that both $p(\boldsymbol{x}|\boldsymbol{c})$ and $p(\boldsymbol{x})$ can be approximated using ELBO:

$$p(\boldsymbol{x}|\boldsymbol{c}) \approx \exp(-\mathbb{E}_{t,\varepsilon}||\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})||^2 + C), \tag{5.2}$$

$$p(\boldsymbol{x}) \approx \exp(-\mathbb{E}_{t,\varepsilon}||\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \varnothing)||^2 + C). \tag{5.3}$$

Notice that the constant C is independent of $\boldsymbol{x}$ and $\boldsymbol{c}$, therefore dividing $p(\boldsymbol{x}|\boldsymbol{c})$ by $p(\boldsymbol{x})$ cancels C. This gives us the approximation of *lift* in the form of $\log \frac{p(\boldsymbol{x}|\boldsymbol{c})}{p(\boldsymbol{x})}$:

$$\mathit{lift}(\boldsymbol{x}|\boldsymbol{c}) \approx \mathbb{E}_{t,\varepsilon}\{||\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \varnothing)||^2 - ||\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})||^2\}. \tag{5.4}$$

Equation 5.4 gives us another explanation of lift scores from a denoising perspective: if $\boldsymbol{c}$ is positively associated with $\boldsymbol{x}$, then given a sample $\boldsymbol{x}_t$ by adding noise to $\boldsymbol{x}$, a diffusion model should improve the reconstruction quality to $\boldsymbol{x}$ from the noisy $\boldsymbol{x}_t$ if $\boldsymbol{c}$ is given to the model, compared to the reconstruction quality of $\boldsymbol{x}_t$ when $\boldsymbol{c}$ is not known. Hence, **the criterion of** *CompLift* for single condition is:

$$\pi(\boldsymbol{x}|\boldsymbol{c}) = \begin{cases} 1 \text{ (accept)} & \text{if Equation 5.4} > 0; \\ 0 \text{ (reject \& resample)} & \text{if Equation 5.4} \leq 0. \end{cases} \tag{5.5}$$

The `Compose` for multiple conditions is straightforward, where we construct a set of

Table 5.1: **Examples of `Compose` for multiple conditions.**

Type	Algebra	Acceptance Criterion
Product	$\boldsymbol{c}_1 \wedge \boldsymbol{c}_2$	$\min_{i\in[1,2]} \mathit{lift}(\boldsymbol{x}\|\boldsymbol{c}_i) > 0$
Mixture	$\boldsymbol{c}_1 \vee \boldsymbol{c}_2$	$\max_{i\in[1,2]} \mathit{lift}(\boldsymbol{x}\|\boldsymbol{c}_i) > 0$
Negation	$\neg \boldsymbol{c}_1$	$\mathit{lift}(\boldsymbol{x}\|\boldsymbol{c}_1) \leq 0$

constraints to satisfy, then compose them (see examples in Table 5.1).

We describe the `Compose` algorithm used in our experiments in Algorithm 9. The algorithm takes as input a set of lift scores $\{\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i)\}_{i=1}^n$ and an algebra $\mathcal{A}$, and returns a boolean value indicating whether the composed prompt is satisfied. The algorithm first initializes a dictionary s to store partial results. It then converts the algebra $\mathcal{A}$ to conjunctive normal form (CNF) and iterates over each disjunction $\mathcal{A}_k$ in $\mathcal{A}$. For each literal $\mathcal{L}_m$ in $\mathcal{A}_k$, the algorithm calculates the lift score based on the input lift scores and updates $s[\mathcal{A}_k]$. Finally, it returns whether the minimum of the results is greater than zero.

Algorithm 9 `Compose` for multiple algebraic operations

Input: lift scores $\{\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i)\}_{i=1}^n$, algebra $\mathcal{A}$
Initialize $s = \{\}$ ▷ Dictionary to store partial results
Convert $\mathcal{A}$ to conjunctive normal form (CNF)
for disjunction $\mathcal{A}_k \in \mathcal{A}$ **do**
    Initialize $s[\mathcal{A}_k] = 0$
    for literal $\mathcal{L}_m \in \mathcal{A}_k$ **do**
        if $\mathcal{L}_m$ is of form $\boldsymbol{c}_i$ **then**
            $s[\mathcal{A}_k] = \max(\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i), s[\mathcal{A}_k])$
        else if $\mathcal{L}_m$ is of form $\neg\boldsymbol{c}_i$ **then**
            $s[\mathcal{A}_k] = \max(-\mathit{lift}(\boldsymbol{x}_0|\boldsymbol{c}_i), s[\mathcal{A}_k])$
return $\min_{\mathcal{A}_k\in\mathcal{A}}(s[\mathcal{A}_k]) > 0$

5.4 Design Space of *CompLift*

We present the naive version of *CompLift* in Algorithm 8, where the lift score in Equation 5.4 is estimated using Monte-Carlo sampling. In this section, we discuss design choices of

CompLift. We wish to answer the following questions:

(Section 5.4.1) How does the noise sampling affect the estimation accuracy of lift scores?

(Section 5.4.2) What role do different timesteps play in the effectiveness of our criterion?

(Section 5.4.3) Can we optimize the computation overhead via caching strategies?

5.4.1 Effect of Noise

In Line 7 in Section 8, we choose to share the ε when estimating both $p(\boldsymbol{x}|\boldsymbol{c})$ and $p(\boldsymbol{x})$. To validate this design, we conduct the experiment on a 2D synthetic dataset with the following 3 settings:

(1) **Independent Noise**: We sample two independent noises ε and ε' for $p(\boldsymbol{x}|\boldsymbol{c})$ and $p(\boldsymbol{x})$, respectively.

(2) **Shared Noise for Each Trial**: We share the noise ε between $p(\boldsymbol{x}|\boldsymbol{c})$ and $p(\boldsymbol{x})$, each trial samples a new noise.

(3) **Shared Noise Across All Trials**: We share the noise ε across all trials for both $p(\boldsymbol{x}|\boldsymbol{c})$ and $p(\boldsymbol{x})$.

In Section 5.2, we observe that sharing noise for each trial is mostly robust over the number of trials, which is consistent with the conclusion in Diffusion Classifier [98]. We choose to use this design as the default setting. The independent strategy increases the accuracy with more trials, especially for mixture and negation algebra. We also find a small regression in the sharing strategies in negation tasks, hypothetically due to the conservativeness of the Lift criterion.

5.4.2 Effect of Timesteps

Similar to Diffusion Classifier [98], we found that in most cases, if diffusion model ε_θ is trained with uniform timestep sampling, sampling the timestep uniformly for ELBO estimation

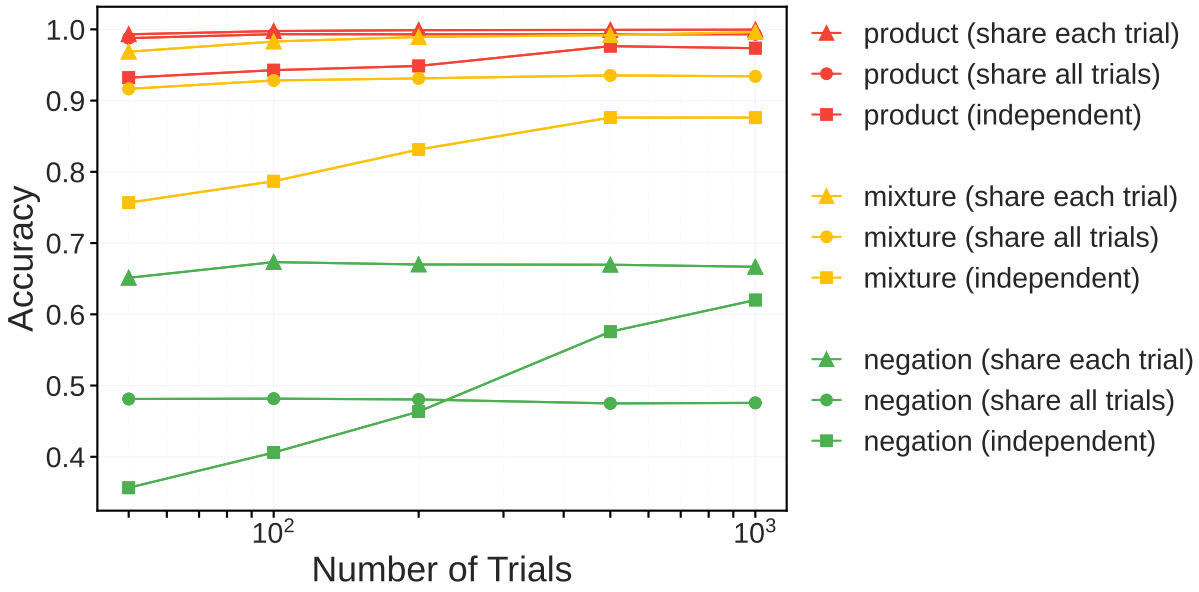


Figure 5.2: **The accuracy of *CompLift* with different noise sampling strategies on 2D synthetic dataset.** See Section 5.4.1.

(Line 4 in Algorithm 8) is sufficient.

More surprisingly, we also found that the timestep for ELBO estimation needs importance sampling when the diffusion model is trained under importance sampling, where the importance sampling enhances the stability of optimizing L_{VLB} [121]. In particular, we discovered a pretrained model for CLEVR Position dataset [105] requires the importance sampling because of the above reason. We show the effect in Figure 5.3.

This suggests that the timestep sampling of ELBO estimation to stay consistent with the sampling strategy at training stage. Therefore, we choose to use importance sampling for the CLEVR pretrained model, and use uniform sampling as default for other tasks, i.e., 2D dataset and text-to-image.

5.4.3 Improving Computational Efficiency with Cached Prediction

In the naive version of *CompLift* in Algorithm 8, the algorithm requires $(n+1)\cdot T$ times of forward passes for n conditions. To reduce the computational cost, we propose caching the

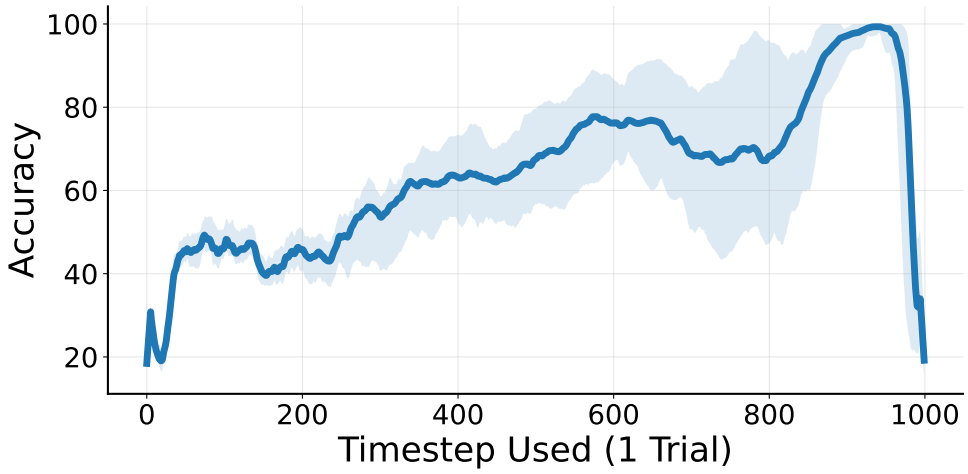


Figure 5.3: **Accuracy of acceptance/rejection over a single sampled timestep for pretrained model [105] on CLEVR Position dataset.** We found that models trained with importance sampling require importance sampling for ELBO estimation.

diffusion model's intermediate predictions.

We find that we can reuse the prediction of $\epsilon_\theta(\boldsymbol{x}_t, \varnothing)$ and $\{\epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c}_i)\}_{i=1}^n$ at intermediate timestep t during the generation process of $\boldsymbol{x}_0$. Note that the unconditional prediction $\epsilon_\theta(\boldsymbol{x}_t, \varnothing)$ is typically precomputed for generation with classifier-free guidance (CFG) [65]. For conditional prediction $\{\epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c}_i)\}_{i=1}^n$, they are precomputed under the Composable Diffusion framework [105]. Under this framework, we can reduce the number of extra forward passes from $(n+1)\cdot T$ to 0.

For more general framework, computing the conditional predictions $\{\epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c}_i)\}_{i=1}^n$ as additional computation on-the-fly during generation is beneficial for caching in Section 5.4.3. With a sufficient amount of GPU memory, this operation typically would not bring extra time overhead, since all computations can be done in parallel to the original generation task. In this context, we can reduce the number of extra forward passes to $n \cdot T$, if CFG is used.

We provide the optimized version of *CompLift* in Algorithm 10. When running Line 4 in Algorithm 8, ϵ is not i.i.d. any more, but computed as $\epsilon = (\boldsymbol{x}_t - \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0)/\sqrt{1-\bar{\alpha}_t}$ instead using the cached $\boldsymbol{x}_t$. Another difference is that T is constrained to be exactly the same as a number of inference steps N. In practice, we found that these 2 changes do not lead to a significant

Algorithm 10 Optimized Rejection with *Cached CompLift*

Input: conditions $\{c_i\}_{i=1}^n$, algebra $\mathcal{A}$, # of reverse timesteps N
Initialize $x_T \sim \mathcal{N}(0, I)$
for timestep $t_j \in [t_N, t_{N-1}, \ldots, t_1]$ **do**
    ▷ Standard generation process
    $x_{t_{j-1}} = \texttt{step}(x_{t_j}, \varepsilon_\theta, \{c_i\}_{i=1}^n)$
    ▷ Add predictions to cache
    if classifier-free guidance **then**
        Add precomputed $\varepsilon_\theta(x_{t_j}, \varnothing)$ to cache
    if Composable Diffusion **then**
        Add precomputed $\{\varepsilon_\theta(x_{t_j}, c_i)\}_{i=1}^n$ to cache
    Add x_{t_j} to cache
run Algorithm 8 with cache, where # of trials $T = N$, and x_{t_j}, $\varepsilon_\theta(x_{t_j}, \varnothing)$, and $\varepsilon_\theta(x_{t_j}, c_i)$ are reused.

performance drop as long as T is not too small. We show the comparison of the running time on 2D synthetic task in Figure 5.5. The overhead introduced by the cached *CompLift* is negligible for the Composable Diffusion baseline [105]. With enough GPU memory, our method can be further optimized to the latency of only 1 forward pass by parallelization, while MCMC methods [41] require sequential computation.

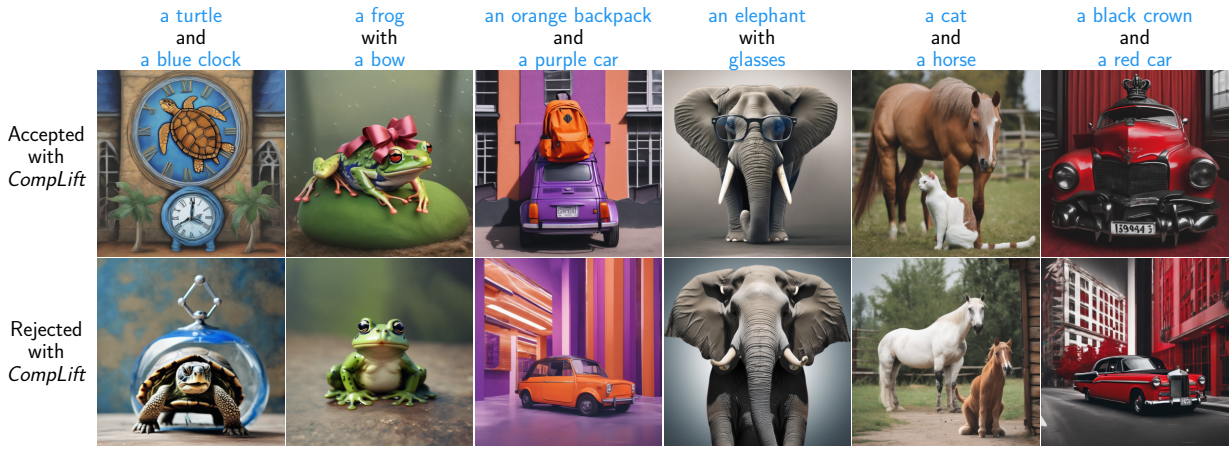


Figure 5.4: **Accepted and rejected SDXL examples using *CompLift* criterion.** Objects in blue are composed through the given prompts.

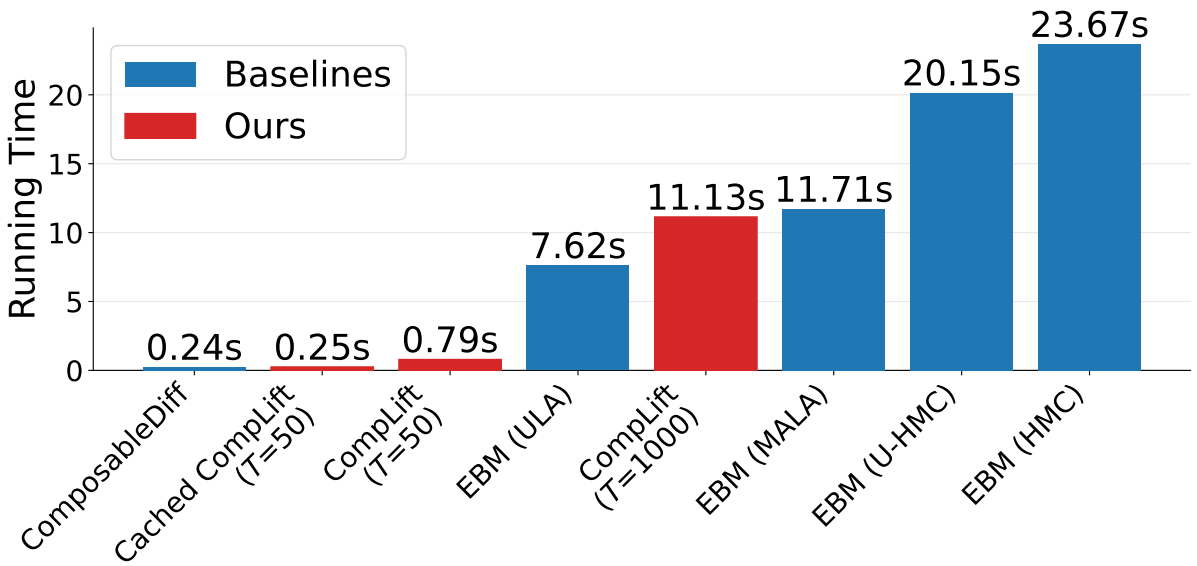


Figure 5.5: **Average running time on 2D synthetic dataset.** T indicates number of trials. Note that our methods can be further optimized to the latency of only 1 forward pass with parallelization, while MCMC methods require sequential computation.

5.5 Scaling to Text-to-Image Generation

We mainly consider the problem of **missing objects** in the text-to-image compositional task. Given multiple objects $\{c_i\}_{i=1}^n$ to address simultaneously, a prominent error of diffusion models is that the generated images sometimes do not include object(s) $\{c_j | j \in [1,n]\}$ mentioned in the text.

5.5.1 Counting Activated Pixels as Existence of Objects

Our approach is to count the number of activated pixels in the latent space to determine the existence of objects. A pixel at position $[a,b]$ in the latent z is activated for condition c_i, if $\mathit{lift}(z,c_i)_{[a,b]} > 0$. Here we regard $\mathit{lift}(z,c_i)$ as a matrix with the same shape of z. This is achieved by averaging only over the timestep t dimension and not the feature dimension when calculating the `mean` in Monte-Carlo estimation, i.e., Line 12 in Figure 8.

Definition of *CompLift* in image space. In particular, an object is considered to exist in the image

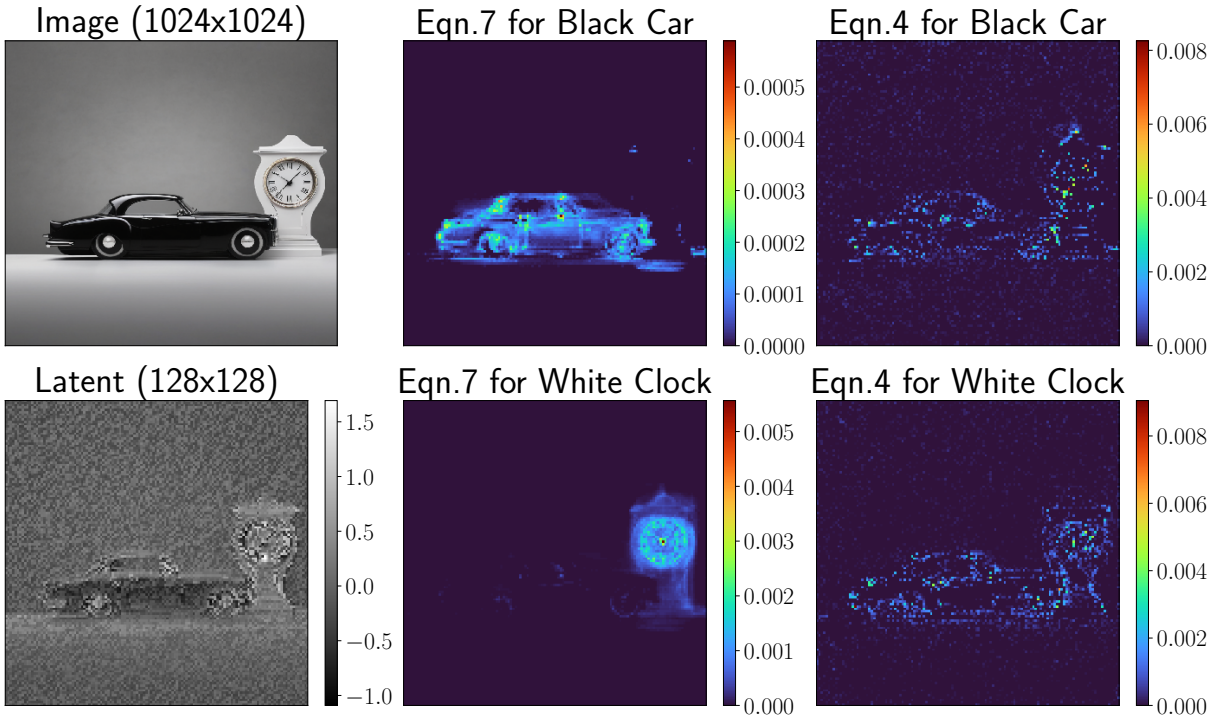


Figure 5.6: **Replacing ε in the differential loss of Equation 5.4 by $\varepsilon_\theta(\boldsymbol{x}, \boldsymbol{c}_{\text{compose}})$ reduces the estimation variance.** Prompt: *a black car and a white clock*. Pixels with negative scores are masked out.

$\boldsymbol{x}$, if the number of activated pixels in its latent $\boldsymbol{z}$ is greater than threshold τ:

$$\mathit{lift}(\boldsymbol{x}, \boldsymbol{c}_i) := \sum_{a,b \in [1,m]} \mathbb{I}(\mathit{lift}(\boldsymbol{z}, \boldsymbol{c}_i)_{[a,b]} > 0) - \tau, \tag{5.6}$$

where $\boldsymbol{z}$ is the latent in the shape of (m, m). τ is the only hyperparameter that needs to be tuned for our approach. We set it as $\tau = 250$ for all experiments.

Equation 5.6 of *CompLift* for images makes the identification of objects more robust and interpretable. The rationale is that the activated pixels in the latent space are more likely to be the ones that correspond to the objects in the image. We present examples in Figure 5.7. As illustrated, we can see a clear consistency between the pixels activated by $\mathit{lift}(\boldsymbol{z}, \boldsymbol{c}_i)$ in the latent and the object $\boldsymbol{c}_i$ in the image. Since the object often occupies only a fraction of the image, simply taking the mean along the feature dimension as Equation 5.4, may not be as effective as Equation

5.6, as those unrelated pixels with $\mathit{lift}(\boldsymbol{z}, \boldsymbol{c}_i) < 0$ will dominate the averaged score.

5.5.2 Reduce Variance of ELBO Estimation

Our initial attempt is to see whether the pixels in the latent $\boldsymbol{z}$ align with the lift score. Intuitively, we expect for an arbitrary pixel with position $[a,b]$ from a target object $\boldsymbol{c}_i$, its lift score $\mathit{lift}(\boldsymbol{z}, \boldsymbol{c}_i)_{[a,b]}$ should be greater than 0 in most cases. However, we found that the estimation variance is high when applying Equation 5.4 directly. As illustrated in the 3rd column in Figure 5.6, the lift score is noisy overall.

We hypothesize that the sampled noise ε in the L2 loss in Equation 5.4 is a major factor of the variance. In the ELBO estimation, we first add the noise to the image and make predictions, similar to adversarial attacks. The model may or may not have a good prediction on the new noisy image, therefore, the model's bias might lead to unstable estimation. To reduce the variance, we can find a candidate to replace ε when calculating the L2 loss in Equation 5.4. Ideally, the candidate is close enough to ε, but inherently cancels out the bias from the model's prediction.

Our finding is that the composed score $\varepsilon_\theta(\boldsymbol{z}, \boldsymbol{c}_{\text{compose}})$ is a good candidate to replace ε, where $\boldsymbol{c}_{\text{compose}}$ is the original composed prompt to generate latent $\boldsymbol{z}$. In the example of Figure 5.6, $\boldsymbol{c}_{\text{compose}}$ is "a black car and a white clock". Since $\varepsilon_\theta(\boldsymbol{z}, \boldsymbol{c}_{\text{compose}})$ mainly guides the generation process of $\boldsymbol{z}$, it is reasonable to assume that it is good at reconstructing the original latent, therefore is close to ε. Furthermore, since $\varepsilon_\theta(\boldsymbol{z}, \boldsymbol{c}_{\text{compose}})$ and $\varepsilon_\theta(\boldsymbol{z}, \boldsymbol{c}_i)$ (or $\varepsilon_\theta(\boldsymbol{z}, \varnothing)$) come from the same model's prediction, it helps cancel the model's bias. The results are shown in the 2nd column of Figure 5.6. We observe that the variance is significantly reduced, and the activated pixels are more consistent with the objects.

The improved lift score for condition $\boldsymbol{c}$ and latent $\boldsymbol{z}$ is:

$$\begin{aligned} \mathit{lift}(\boldsymbol{z}, \boldsymbol{c}) := \mathbb{E}_{t,\varepsilon}\{ &||\varepsilon_\theta(\boldsymbol{z}_t, \boldsymbol{c}_{\text{compose}}) - \varepsilon_\theta(\boldsymbol{z}_t, \varnothing)||^2 \\ &- ||\varepsilon_\theta(\boldsymbol{z}_t, \boldsymbol{c}_{\text{compose}}) - \varepsilon_\theta(\boldsymbol{z}_t, \boldsymbol{c})||^2\} \end{aligned} \tag{5.7}$$

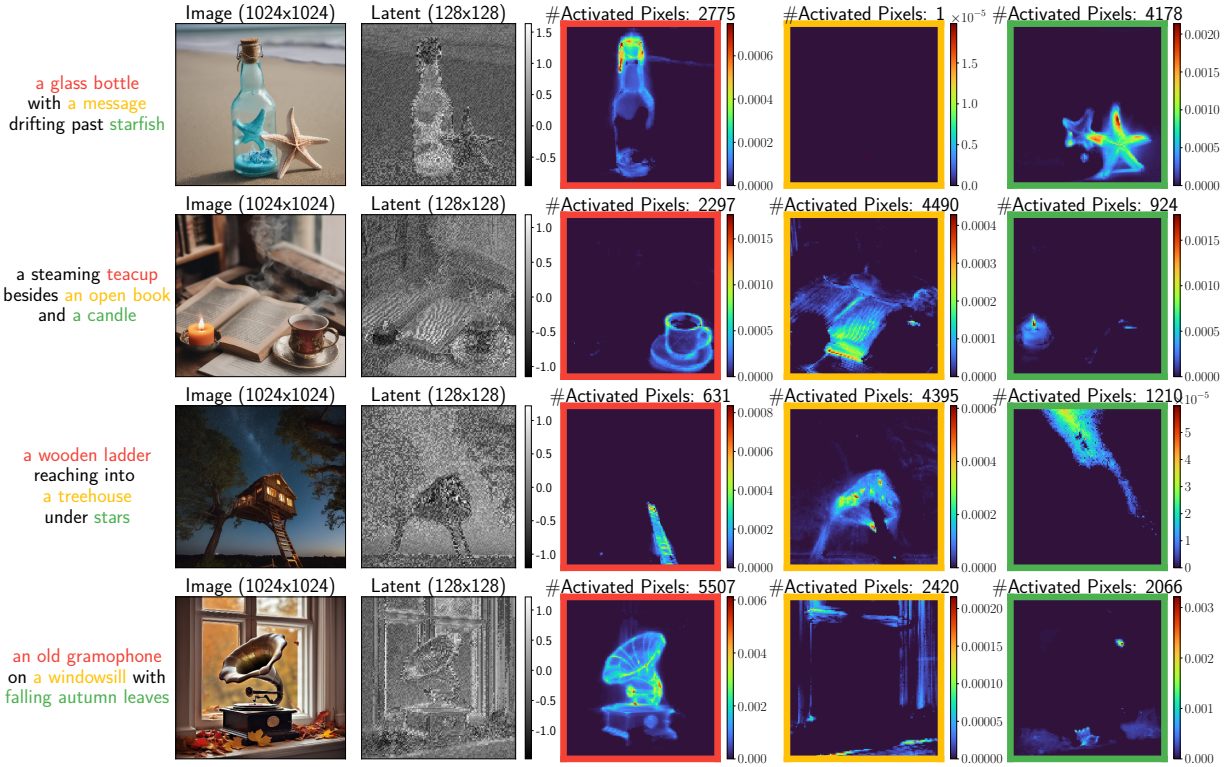


Figure 5.7: *CompLift* **for text-to-image compositional task.** Pixels with negative scores are masked out. From left to right: original image $\boldsymbol{x}$, the latent $\boldsymbol{z}$, the heatmaps of lift score $\mathit{lift}(\boldsymbol{z}, \boldsymbol{c}_i)$ for each object component $\boldsymbol{c}_i$. The border color represents the corresponding object.

The combination of Equation 5.6 and Equation 5.7 represents our final criterion for text-to-image compositional task. We first estimate the lift score for each pixel in the latent space via Equation 5.7, and then count the activated pixels to determine the existence of objects in the image via Equation 5.6. Note that for the improved lift score, the computational overhead is $(n+2)\cdot T$ forward passes, if no caching or parallelization is used.

5.5.3 Full Algorithms

We combine Algorithm 8, Equation 5.6, and Equation 5.7, and provide the full algorithm of *CompLift* for t2i tasks.

Similar to the vanilla version, we combine Algorithm 10, Equation 5.6, and Equation 5.7, and provide the full algorithm of *Cached CompLift* for t2i tasks. The algorithm is divided into

Algorithm 11 Text-to-Image Generation with CompLift

Input: image $\boldsymbol{x}_0$, conditions $\{\boldsymbol{c}_i\}_{i=1}^n$, composed prompt $\boldsymbol{c}_{\text{compose}}$, # of trials T, threshold τ
Initialize $\texttt{PixelLift}[\boldsymbol{c}_i] = \text{list}()$ for each $\boldsymbol{c}_i$
Encode $\boldsymbol{x}_0$ to latent $\boldsymbol{z}_0$ using VAE encoder
for trial $j = 1, \ldots, T$ **do**
  Sample $t \sim [1, 1000]$; $\varepsilon \sim \mathcal{N}(0, I)$
  $\boldsymbol{z}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{z}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$
  $\varepsilon_{\text{compose}} = \varepsilon_\theta(\boldsymbol{z}_t, \boldsymbol{c}_{\text{compose}})$ ▷ Get composed prediction
  for condition $\boldsymbol{c}_k \in \{\boldsymbol{c}_i\}_{i=1}^n$ **do**
    $lift_j(\boldsymbol{z}_0|\boldsymbol{c}_k) = \|\varepsilon_{\text{compose}} - \varepsilon_\theta(\boldsymbol{z}_t, \varnothing)\|^2 - \|\varepsilon_{\text{compose}} - \varepsilon_\theta(\boldsymbol{z}_t, \boldsymbol{c}_k)\|^2$
    $\texttt{PixelLift}[\boldsymbol{c}_k].\texttt{append}(lift_j(\boldsymbol{z}_0|\boldsymbol{c}_k))$
for condition $\boldsymbol{c}_k \in \{\boldsymbol{c}_i\}_{i=1}^n$ **do**
  $lift(\boldsymbol{z}_0, \boldsymbol{c}_k) = \texttt{mean}(\texttt{PixelLift}[\boldsymbol{c}_k])$ ▷ Average over trials
  $lift(\boldsymbol{x}_0, \boldsymbol{c}_k) = \sum_{a,b \in [1,m]} \mathbb{I}(lift(\boldsymbol{z}_0, \boldsymbol{c}_k)_{[a,b]} > 0) - \tau$
return $\texttt{Compose}(\{lift(\boldsymbol{x}_0|\boldsymbol{c}_i)\}_{i=1}^n, \mathcal{A})$ ▷ Run Algorithm 9

two parts: Algorithm 12 and Algorithm 13. The first part caches the latent vectors and predictions at each timestep, while the second part calculates the lift scores using the cached values.

5.6 Experiments

In the following sections, we evaluate the effectiveness of our *CompLift* criterion on 2D synthetic dataset, CLEVR Position dataset, and text-to-image compositional task.

5.6.1 2D Synthetic Dataset

Experiment setup. We train diffusion models on 2D synthetic dataset with 3 types of compositional algebra: product, mixture, and negation. We use the same architecture of networks from [41], which regards the diffusion model ε_θ as $\nabla_{\boldsymbol{x}} E_\theta$. E_θ is a 3-layer MLP and trained with the same loss function as the standard diffusion model by minimizing $\mathbb{E}_{t,\varepsilon}||\nabla_{\boldsymbol{x}} E_\theta(\boldsymbol{x}_t, t) - \varepsilon||_2^2$. An individual model is trained for each single distribution with 8000 samples. We use the diffusion process of 50 timesteps for both training and inference, which means the cached version of

Algorithm 12 Cache Values During Text-to-Image Generation

Input: conditions $\{c_i\}_{i=1}^n$, composed prompt c_{compose}, # of timesteps N
Initialize $z_T \sim \mathcal{N}(0, I)$
Initialize `Cache` $= \{\}$ ▷ Dictionary to store intermediate values
for timestep $t_j \in [t_N, t_{N-1}, \ldots, t_1]$ **do**
  ▷ Standard generation process
  $z_{t_{j-1}} = \texttt{step}(z_{t_j}, \varepsilon_\theta, c_{\text{compose}})$
  ▷ Add predictions to cache
  $\texttt{Cache}[t_j].\texttt{latent} = z_{t_j}$
  $\texttt{Cache}[t_j].\texttt{null\_pred} = \varepsilon_\theta(z_{t_j}, \varnothing)$
  for condition $c_k \in \{c_i\}_{i=1}^n$ **do**
    $\texttt{Cache}[t_j].\texttt{cond\_pred}[c_k] = \varepsilon_\theta(z_{t_j}, c_k)$
  $\texttt{Cache}[t_j].\texttt{compose\_pred} = \varepsilon_\theta(z_{t_j}, c_{\text{compose}})$
Decode z_0 to image x_0 using VAE decoder
return x_0, `Cache`

Algorithm 13 CompLift Using Cached Values

Input: image x_0, conditions $\{c_i\}_{i=1}^n$, `Cache`, threshold τ
Initialize $\texttt{PixelLift}[c_i] = \text{list}()$ for each c_i
Encode x_0 to latent z_0 using VAE encoder
for timestep t_j in `Cache` **do**
  $z_{t_j} = \texttt{Cache}[t_j].\texttt{latent}$
  $\varepsilon_{\text{compose}} = \texttt{Cache}[t_j].\texttt{compose\_pred}$
  for condition $c_k \in \{c_i\}_{i=1}^n$ **do**
    $lift_j(z_0|c_k) = \|\varepsilon_{\text{compose}} - \texttt{Cache}[t_j].\texttt{null\_pred}\|^2 - \|\varepsilon_{\text{compose}} - \texttt{Cache}[t_j].\texttt{cond\_pred}[c_k]\|^2$
    $\texttt{PixelLift}[c_k].\texttt{append}(lift_j(z_0|c_k))$
for condition $c_k \in \{c_i\}_{i=1}^n$ **do**
  $lift(z_0, c_k) = \texttt{mean}(\texttt{PixelLift}[c_k])$ ▷ Average over timesteps
  $lift(x_0, c_k) = \sum_{a,b \in [1,m]} \mathbb{I}(lift(z_0, c_k)_{[a,b]} > 0) - \tau$
return $\texttt{Compose}(\{lift(x_0|c_i)\}_{i=1}^n, \mathcal{A})$ ▷ Run Algorithm 9

Table 5.2: Quantitative results on 2D compositional generation. Acc (%) means accuracy, and CD means Chamfer Distance.

Method	**Product**		**Mixture**		**Negation**	
	Acc ↑	CD ↓	Acc ↑	CD ↓	Acc ↑	CD ↓
Composable Diffusion	56.5	0.061	70.7	0.042	7.9	0.207
EBM (ULA)	51.7	0.076	71.8	0.044	11.4	0.186
EBM (U-HMC)	56.3	0.036	73.1	0.063	9.4	0.269
EBM (MALA)	62.6	0.054	73.8	0.036	6.8	0.203
EBM (HMC)	64.8	0.021	73.7	0.078	4.2	0.340
Composable Diffusion + *Cached CompLift*	99.5	0.009	86.5	0.023	62.0	0.137
Composable Diffusion + *CompLift* ($T = 50$)	99.1	0.015	98.6	0.029	75.0	0.154
Composable Diffusion + *CompLift* ($T = 1000$)	**99.9**	**0.009**	**100.0**	**0.023**	**78.7**	**0.131**

CompLift uses 50 trials. The vanilla version of *CompLift* uses 50 or 1000 trials. All the methods generate 8000 samples for inference. For *CompLift* algorithms, we use Composable Diffusion [105] to generate the initial 8000 samples, then apply the *CompLift* criterion to accept or reject samples. We include MCMC methods as additional baselines [41].

Metrics. We define that a sample satisfies a uniform distribution if it falls into the nonzero-density regions. A sample satisfies Gaussian mixtures if it is within 3σ of any Gaussian component, where σ is the standard deviation of the component. The negation condition is satisfied, if the above description is not satisfied. We calculate the accuracy of the algorithm on a composed distribtuion, as the percentage of generated samples that satisfy all the conditions. We use the Chamfer Distance to measure the similarity between the generated samples and the dataset, as we find it an efficient and general metric, applicable to all 3 kinds of algebra.

It is genuinely hard to define a universal $\varepsilon_\theta(\boldsymbol{x}_t, \varnothing)$ for the 2D synthetic task, since in this task, the 2D distribution can be of arbitrary shapes. Therefore, we find an effective strategy by defining $\varepsilon_\theta(\boldsymbol{x}_t, \varnothing) := \alpha\varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})$ with $\alpha = 0.9$ for each $\boldsymbol{c}$. Consequently, the *lift* criterion is simplified as $\mathbb{E}_{t,\varepsilon}\{||\varepsilon - \alpha\varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})||^2 - ||\varepsilon - \varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})||^2\}$. We find this criterion is surprisingly effective for this 2D task. One hypothesis is that if $\boldsymbol{x}$ has some correlation with $\boldsymbol{c}$, then a well-trained ε_θ should be good at predicting the noise $\varepsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c})$ that is closer to ε and further from $\mathbf{0}$ in most cases.

The overall result is shown in Table 5.2. We observe that the *CompLift* criterion is effective in all 3 types of compositional algebra. Compared to MCMC methods, rejection with *CompLift* criterion is able to generate empty sets by construction.

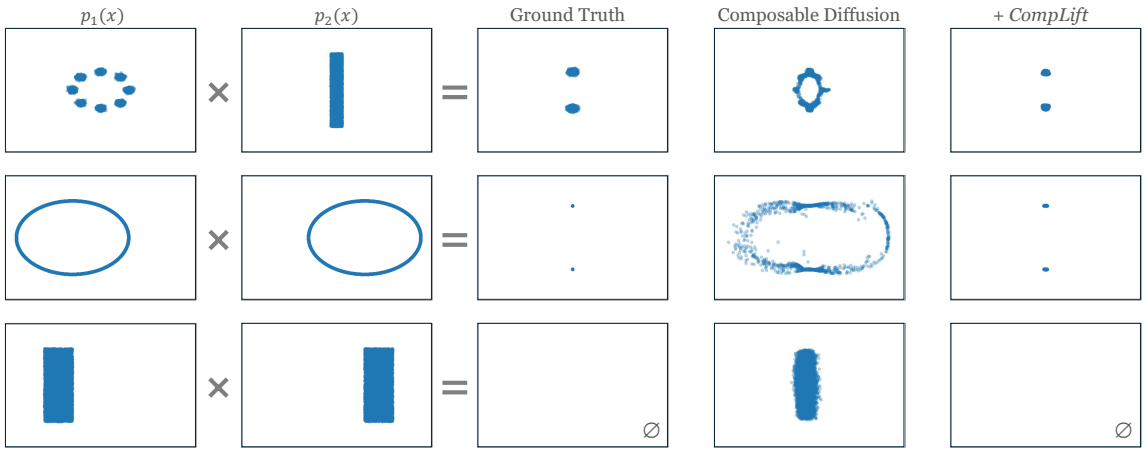


Figure 5.8: 2D synthetic result for product composition.

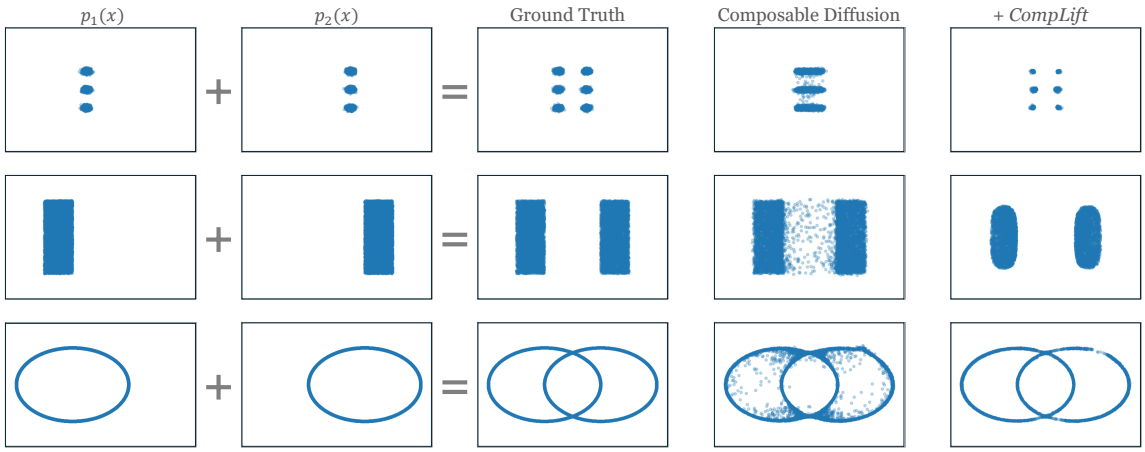


Figure 5.9: 2D synthetic result for mixture composition.

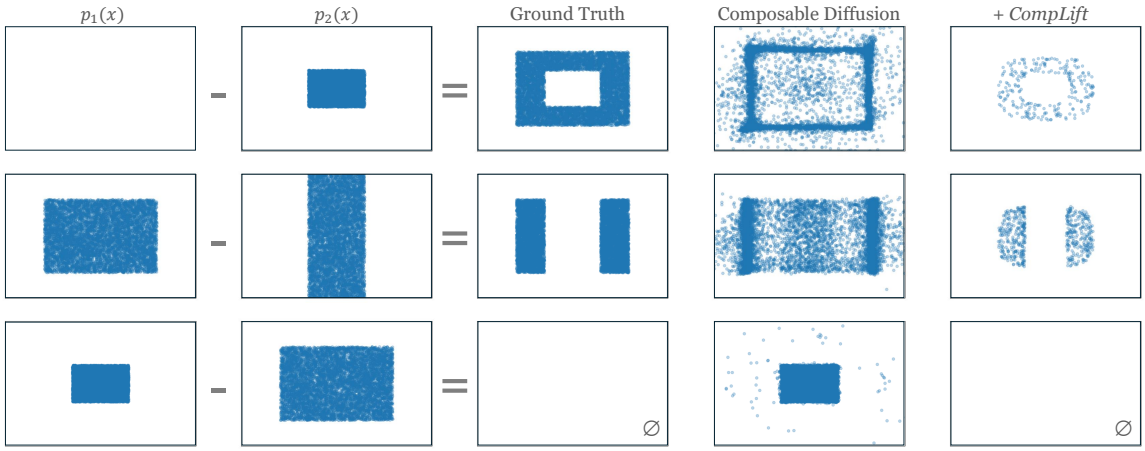


Figure 5.10: 2D synthetic result for negation composition.

In Figures 5.8, 5.9, and 5.10, we provide additional visualizations of the 2D synthetic distribution compositions. The figures show the generated samples for product, mixture, and negation compositions, respectively.

Table 5.3 shows the detailed results for the 2D synthetic distribution compositions. We provide the accuracy (Acc), Chamfer distance (CD), and acceptance ratio (Ratio) of samples using the *CompLift* algorithm for each composition. We find that Composable Diffusion enhanced with *CompLift* ($T = 1000$) consistently outperforms baseline methods across all scenarios, achieving perfect or near-perfect accuracy in many cases. While all methods perform reasonably well with Product algebra, performance generally degrades with Mixture and especially with Negation

Table 5.3: Quantitative results on 2D composition. CompDiff: Composable Diffusion, S1-S3: Scenarios 1-3. All the CompLift criterion are applied to the CompDiff samples.

Algebra	**Method**	**S1**			**S2**			**S3**		
		Acc ↑	CD ↓	Ratio	Acc ↑	CD ↓	Ratio	Acc ↑	CD ↓	Ratio
Product	EBM (ULA)	80.3	0.010	-	74.7	0.141	-	0.0	-	-
	EBM (U-HMC)	84.9	0.007	-	83.9	0.065	-	0.0	-	-
	EBM (MALA)	95.8	0.005	-	92.0	0.103	-	0.0	-	-
	EBM (HMC)	95.7	**0.003**	-	98.8	0.039	-	0.0	-	-
	CompDiff	81.9	0.030	-	87.6	0.093	-	0.0	-	-
	+ *Cached CompLift*	98.5	0.005	-	**100.0**	0.013	-	**100.0**	-	-
	+ *CompLift* (T=50)	97.4	0.011	-	**100.0**	0.019	-	**100.0**	-	-
	+ *CompLift* (T=1K)	**99.8**	0.005	79.3	**100.0**	**0.013**	55.1	**100.0**	-	0.0
Mixture	EBM (ULA)	28.2	0.062	-	87.9	0.021	-	99.2	0.049	-
	EBM (U-HMC)	30.5	0.060	-	89.5	0.020	-	99.5	0.108	-
	EBM (MALA)	28.4	0.059	-	92.9	**0.018**	-	**100.0**	0.030	-
	EBM (HMC)	28.7	0.059	-	92.3	0.019	-	**100.0**	0.157	-
	CompDiff	22.1	0.067	-	90.0	0.025	-	99.9	0.033	-
	+ *Cached CompLift*	61.0	0.030	-	98.5	0.022	-	**100.0**	0.018	-
	+ *CompLift* (T=50)	95.7	0.020	-	**100.0**	0.041	-	**100.0**	0.026	-
	+ *CompLift* (T=1K)	**100.0**	**0.017**	9.2	**100.0**	0.038	56.8	**100.0**	**0.014**	57.1
Negation	EBM (ULA)	28.7	0.129	-	5.6	0.244	-	0.0	-	-
	EBM (U-HMC)	25.9	0.206	-	2.3	0.333	-	0.0	-	-
	EBM (MALA)	17.8	0.150	-	2.7	0.257	-	0.0	-	-
	EBM (HMC)	12.4	0.258	-	0.2	0.421	-	0.0	-	-
	CompDiff	11.8	0.191	-	11.7	0.222	-	0.0	-	-
	+ *Cached CompLift*	40.4	0.152	-	45.6	**0.123**	-	**100.0**	-	-
	+ *CompLift* (T=50)	68.6	0.150	-	56.4	0.158	-	**100.0**	-	-
	+ *CompLift* (T=1K)	**78.7**	**0.127**	3.2	**57.2**	0.135	6.3	**100.0**	-	0.0

algebra, where *CompLift* still maintains a significant edge over other approaches. The results demonstrate that *CompLift* effectively improves both accuracy and sample quality (measured by Chamfer Distance), though this comes with varying sampling efficiency as shown by the acceptance ratios.

Figure 5.11 shows the histograms of the *CompLift* scores for each *composed* 2D synthetic distributions following the order in the Figure 5.8, Figure 5.9, and Figure 5.10, where the groud-truth positive data points are in color blue, and the negative data points are in color orange. We observe that ❶ positive data samples have a higher *CompLift* score and negative data samples (i.e.,

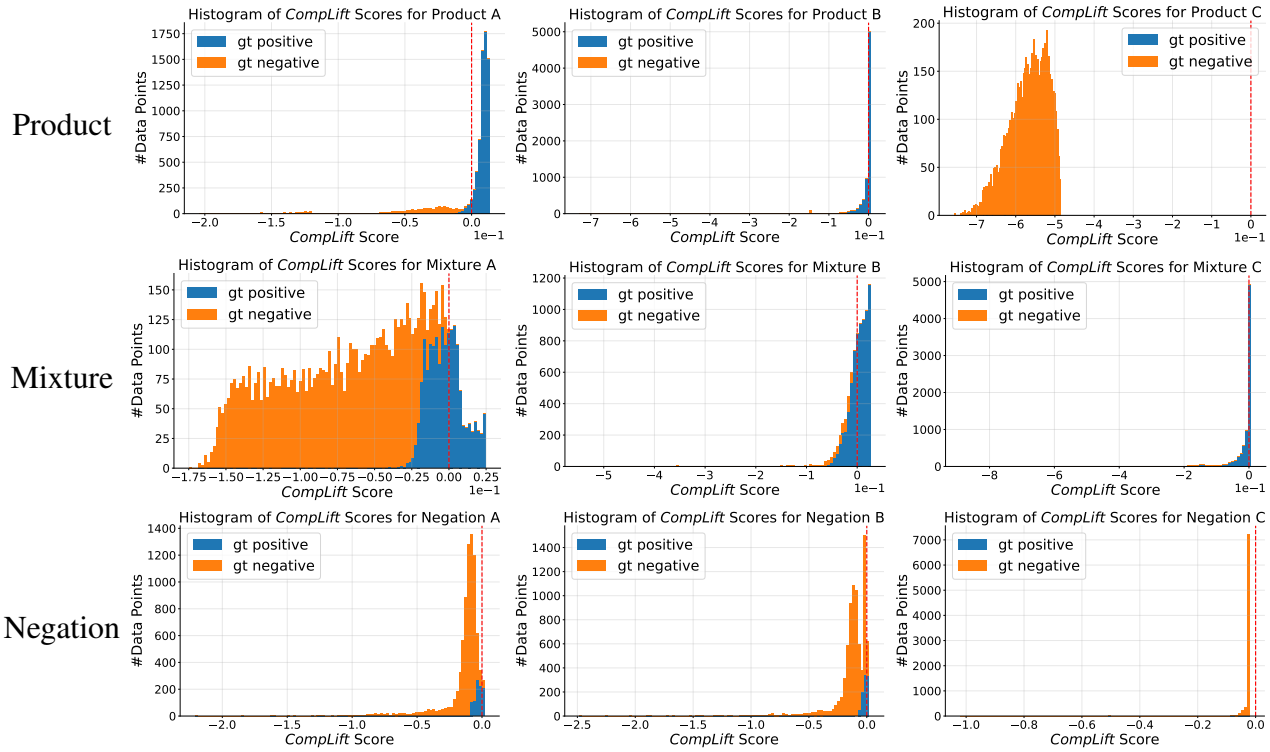


Figure 5.11: **Histograms of the *CompLift* scores for 2D synthetic distribution compositions.** The histograms show the distribution of the lift scores for the three compositions. The samples are 8000 samples generated from Composable Diffusion. The x-axis represents the composed lift scores using Table 5.1, and the y-axis represents the number of samples.

the data sample out of the desired distribution) have a lower *CompLift* score, and ❷ the boundary condition of 0 is a good threshold to separate the positive and negative data samples for product. For mixture and negation, the 0 boundary condition is less effective and slightly conservative, but the positive data samples still have a higher *CompLift* score than the negative data samples, and the performance still achieves better when applying *CompLift* to the baseline methods.

5.6.2 CLEVR Position Dataset

Experiment setup. The CLEVR Position dataset [76] is a dataset of rendered images with a variety of objects placed in different positions. Given a set of 1-5 specified positions, the model needs to generate an image that contains all objects in the specified positions (i.e., product algebra). We use the pretrained diffusion model from [105]. The model is trained with importance

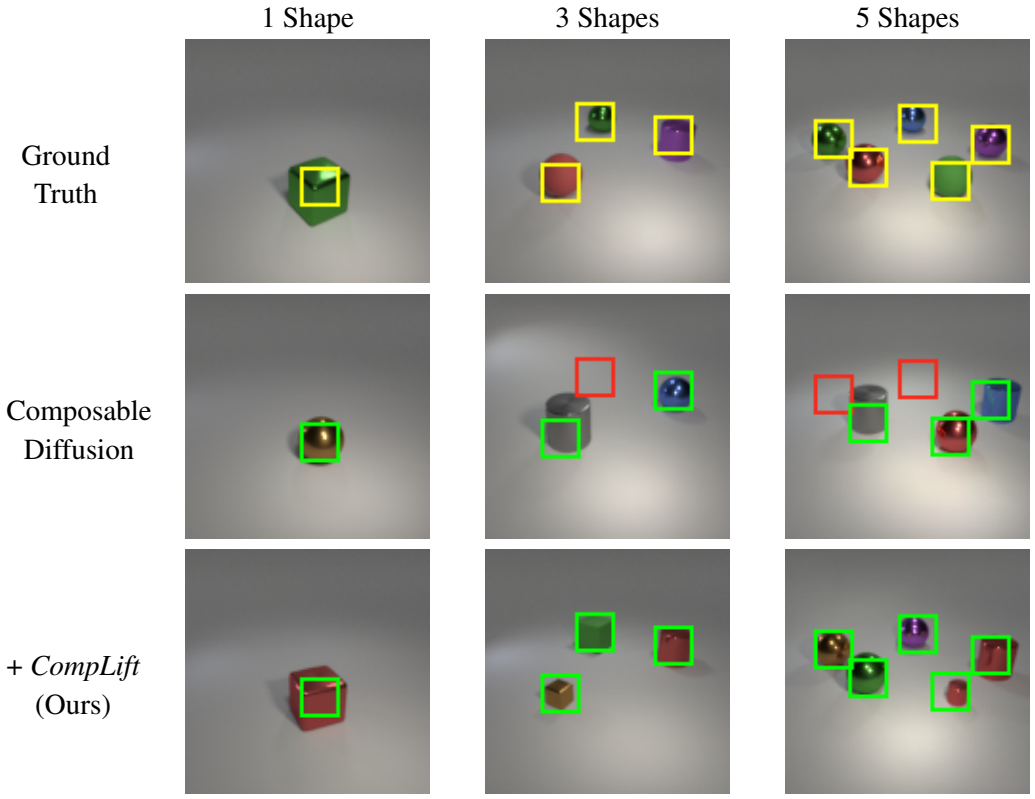


Figure 5.12: **Examples of samples on CLEVR Position dataset.**

sampling [121]. Thus, we use importance sampling for ELBO estimation. In particular, we find that sampling t solely as $t = 928$ is sufficient to give us a decent performance by observing Figure 5.3. This leads to a computational efficiency via only 1 trial at $t = 928$ for both vanilla and cached versions of *CompLift*. For each composition of conditions, our algorithm uses Composable Diffusion [105] to generate the initial 10 samples, then apply the *Lift* criterion to accept or reject samples. We test all methods with 5000 combinations of positions with various numbers of constraints. All methods use classifier-free guidance with a guidance scale of 7.5 [65].

For the MCMC implementation, we use the same PyTorch samplers shared by Du et al. [41] [1]. However, we find it hard to reproduce the result of the MCMC method on the CLEVR Position Task. The generated objects in the synthesized images are often in strange shapes that are dissimilar from the trained data. We suspect that the MCMC method may require more careful

[1] https://github.com/yilundu/reduce_reuse_recycle/blob/main/anneal_samplers.py

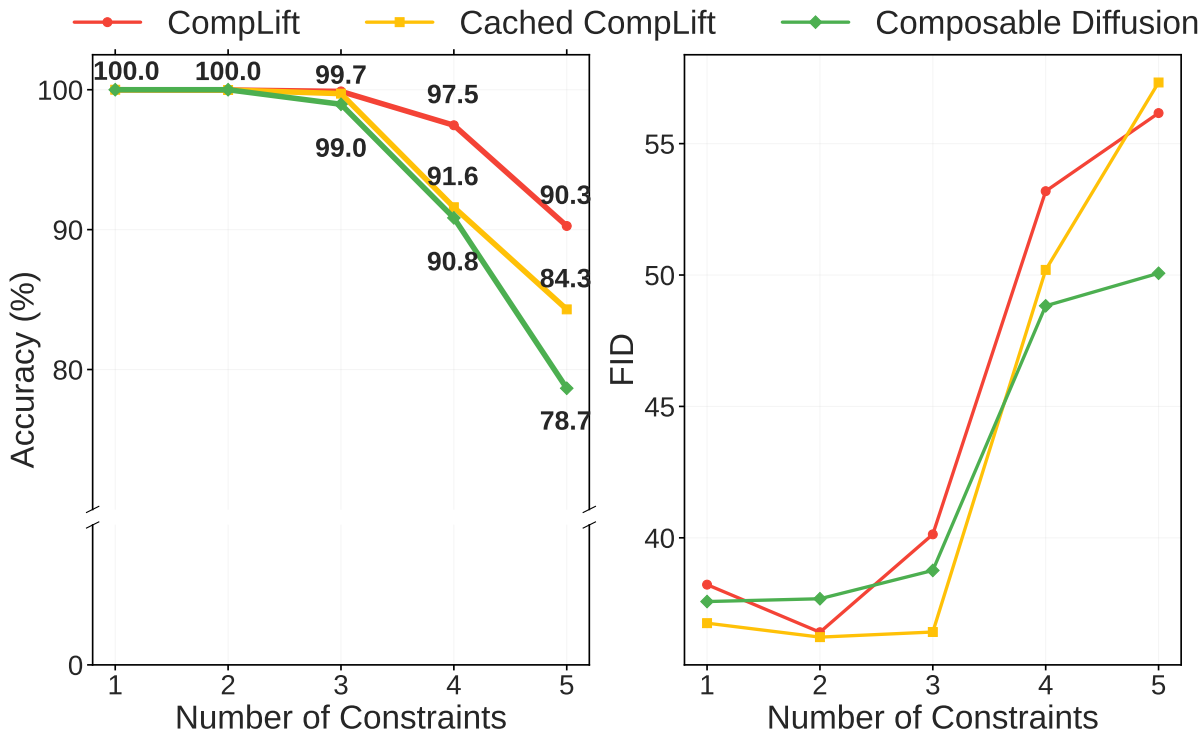


Figure 5.13: **Quantitative results on CLEVR Position dataset.**

tuning of hyperparameters, as not all hyperparameters are mentioned as the optimal ones in the codebase. *Therefore, we remind the readers that the results of the MCMC method on the CLEVR Position Task in this section should be interpreted with caution - they mainly serve as references and do not represent the optimal results.*

Metrics. We use Segment Anything Model 2 (SAM2) [137] as a generalizable verifier, to check whether an object is in a specified position. We first extract the background mask, which is the mask with the largest area, using the automatic mask generator of SAM2, then label a condition as satisfied if the targeted position is not in the background mask. We find this new verifier is more robust compared to the pretrained classifier in Liu et al. [105]. As shown in Figure 5.15, the pretrained classifier often fails to generalize when more shapes are required to be composed, while the SAM2 verifier can effectively detect the samples due to the fact that it is pretrained with massive segmentation dataset. We calculate the accuracy as the ratio of samples that satisfy all given conditions. To calculate FID, we compare the generated samples to the training set

provided by Liu et al. [105].

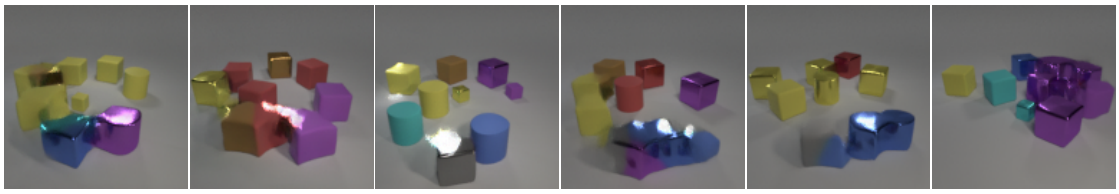


Figure 5.14: **Example of failed samples on CLEVR Position dataset using MCMC.** The generated objects in the synthesized images are often in strange shapes which is dissimilar from the trained data. We suspect that the MCMC method may require a more careful tuning of hyperparameters. As a result, we remind the readers that the results of the MCMC method on the CLEVR Position Task should be interpreted with caution.

We show the quantitative results in Figure 5.13. We observe that the *CompLift* criterion is able to generate samples that satisfy all conditions more effectively than the baseline. As the number of constraints increases, the performance of the baseline drops significantly, while the performance of the *CompLift* criterion remains relatively stable. We also show some examples in Figure 5.12. Meanwhile, we observe some regressions in the FID scores. We hypothesize that the rejection reduces the diversity of samples, which results in the higher FID scores, since the image quality has no significant difference from Composable Diffusion.

Table 5.4: Quantitative accuracy results on CLEVR compositional generation (higher is better).

Method	**Combinations**				
	1	2	3	4	5
Composable Diffusion	**100.0**	**100.0**	99.0	90.8	78.7
EBM (U-HMC)	82.0	66.0	34.0	24.0	11.0
EBM (ULA)	86.0	74.0	48.0	27.0	19.0
Composable Diffusion + *Cached CompLift*	**100.0**	**100.0**	99.7	91.6	84.3
Composable Diffusion + *CompLift*	**100.0**	**100.0**	**99.9**	**97.5**	**90.3**

Nevertheless, we are still interested in how the methods perform using the pretrained classifier from Liu et al. [105]. In Table 5.6, we find that the Composable Diffusion model with *CompLift* consistently outperforms the baseline methods across all combinations, and achieves a nonzero accuracy in the most challenging 5 object composition setting.

Table 5.5: FID scores on CLEVR compositional generation (lower is better).

Method	Combinations				
	1	2	3	4	5
Composable Diffusion	37.6	37.7	38.8	**48.8**	**50.1**
EBM (U-HMC)	158.9	139.7	111.3	101.4	84.4
EBM (ULA)	180.9	153.1	122.4	103.9	84.5
Composable Diffusion + *Cached CompLift*	**36.8**	**36.2**	**36.4**	50.2	57.3
Composable Diffusion + *CompLift*	38.2	36.4	40.1	53.2	56.2

Table 5.6: **Quantitative accuracy results on CLEVR using the pretrained classifier from Liu et al. [105]**.

Method	Combinations				
	1	2	3	4	5
Composable Diffusion	52.7	20.5	3.1	0.3	0.0
EBM (U-HMC)	1.0	0.0	0.0	0.0	0.0
EBM (ULA)	6.0	0.0	0.0	0.0	0.0
Composable Diffusion + *Cached CompLift*	55.7	**25.3**	**6.7**	0.4	0.0
Composable Diffusion + *CompLift*	**56.2**	**25.3**	5.7	**0.7**	**0.1**

5.6.3 Text-to-Image Compositional Task

Experiment setup. We use the pretrained Stable Diffusion 1.4, 2.1, and SDXL [141, 128]. The models are trained with large-scale text-to-image dataset LAION [148]. We use the same composed prompts as Attend-and-Excite Benchmark [24], which considers 3 classes of compositions - 2 animal categories, 1 animal and 1 object category, and 2 object categories (i.e., product algebra). There are 66 unique animal combinations, 66 unique object combinations, and 144 unique animal-object combinations. We use the diffusion model to generate 5 initial samples using each combined prompt, then apply the *CompLift* criterion to accept or reject samples. For inference, we use 50 diffusion timesteps, which means the cached version of *CompLift* uses 50 trials. The vanilla version of *CompLift* uses 200 trials. We use CFG with a guidance scale of 7.5 for all methods. We fix $\tau = 250$ for all experiments.

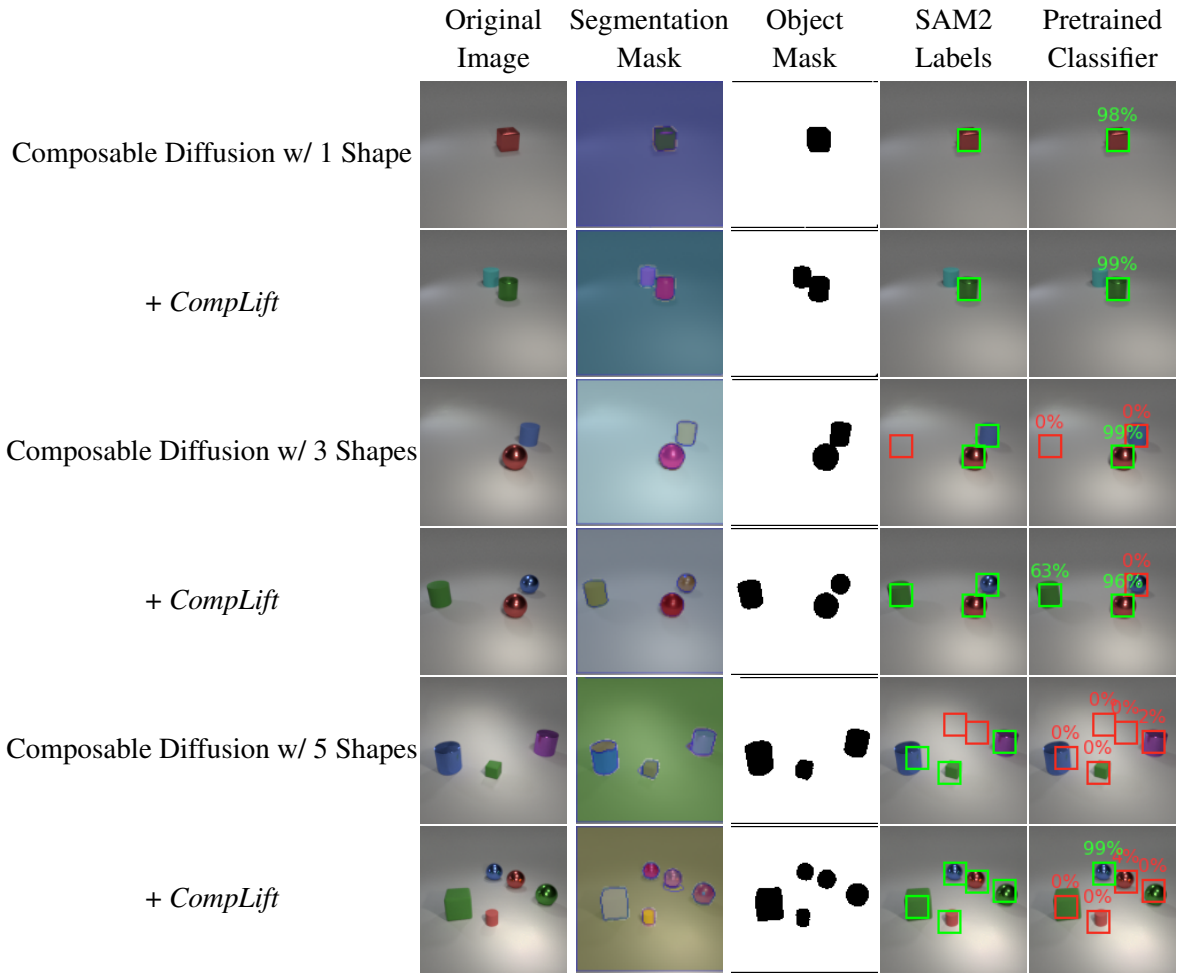


Figure 5.15: **More examples of samples on CLEVR Position dataset.** We find that the SAM verifier is more robust and generalizable compared to the pretrained classifier provided by the Composable Diffusion work [105]. From left to right: the first column shows the original images, the second column shows the segmentation masks generated by SAM2, the third column shows the object masks (by excluding the pixels from the background mask), the fourth column shows the labels given by SAM2, and the fifth column shows the labels given by the pretrained classifier. The percentage numbers represent the confidence score predicted by the classifier.

Metrics. We use CLIP [134] and ImageResizer [137] to assess the alignment between the generated images and the prompts. For combination where *CompLift* is invalid for all generated samples, we uniformly sample one of the images as the final output. The metrics are averaged over the samples within each combination class.

The quantitative results are shown in Table 5.7. We observe that with *CompLift*, the performance for each version of SD can be boosted to be comparable to their next generation for

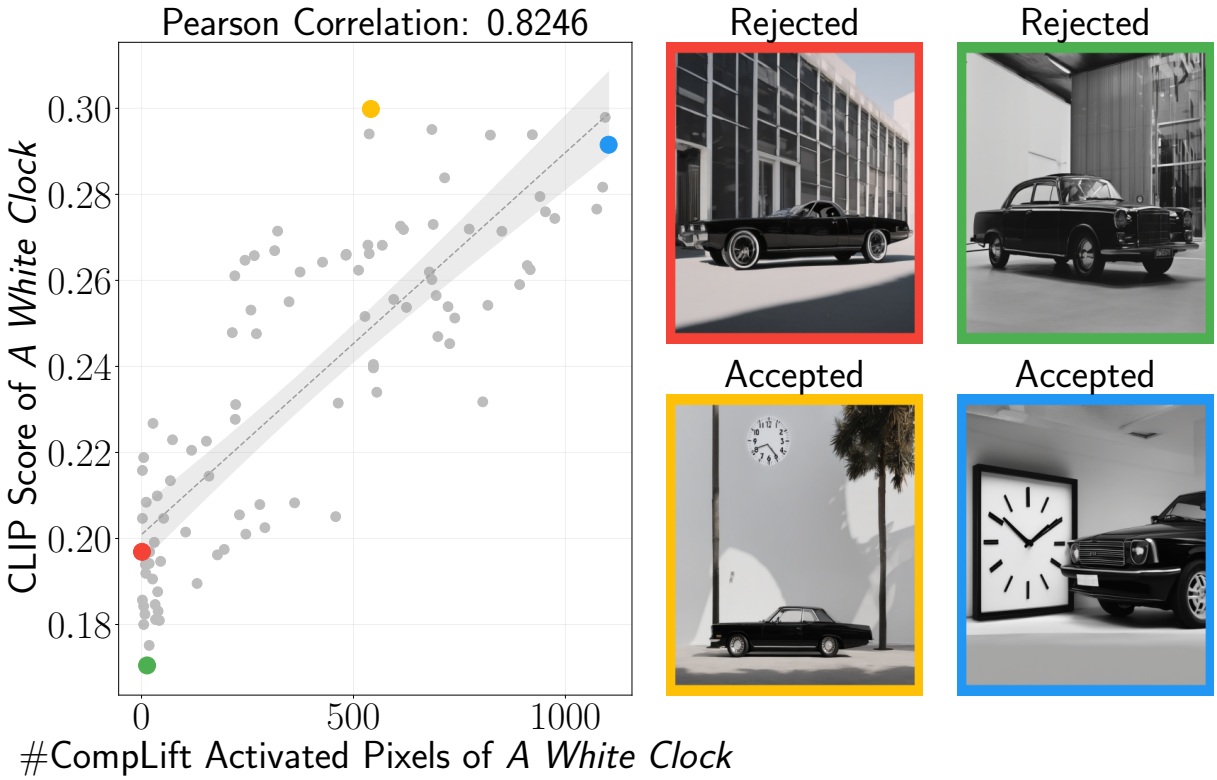


Figure 5.16: **The correlation between the number of activated pixels and the CLIP score.** The correlation is calculated using the Pearson correlation coefficient. Each point represents a generated image. Prompt: *a black car and a white clock*.

some combination classes. For example, SD 1.4 with *CompLift* is comparable to the vanilla SD 2.1 on object-animal combinations and object combinations. SD 2.1 with *CompLift* is comparable to the vanilla SDXL on animal combinations. Since the *CompLift* criterion is a training-free criterion and does not depend on additional verifier, it also shows a novel way of how diffusion model could self-improve during test time without extra training cost [111].

In Figure 5.18 and Figure 5.19, we provide additional examples of accepted and rejected images using SDXL + *CompLift*. We observe that the *CompLift* algorithm can effectively reject samples that fail to capture the object components specified in the prompt in general. In addition, we find that the current *CompLift* criterion is relatively weak in determining the color attribute, leading to some accepted images with incorrect colors (e.g., *a green backpack and a purple bench*). This suggests that future work could explore more sophisticated criteria to further improve the

Table 5.7: Quantitative results on Attend-and-Excite Benchmarks [24]. IR means ImageReward [176].

Method	**Animals**		**Object&Animal**		**Objects**	
	CLIP ↑	IR ↑	CLIP ↑	IR ↑	CLIP ↑	IR ↑
Stable Diffusion 1.4	0.310	-0.191	0.343	0.432	0.333	-0.684
SD 1.4 + *Cached CompLift*	0.319	0.128	0.356	0.990	0.345	-0.124
SD 1.4 + *CompLift*	**0.322**	**0.292**	**0.358**	**1.094**	**0.347**	**-0.050**
Stable Diffusion 2.1	0.330	0.532	0.354	0.924	0.342	-0.112
SD 2.1 + *Cached CompLift*	0.339	0.880	0.361	1.252	0.354	0.353
SD 2.1 + *CompLift*	**0.340**	**0.975**	**0.362**	**1.283**	**0.355**	**0.489**
Stable Diffusion XL	0.338	1.025	0.363	1.621	0.359	0.662
SD XL + *Cached CompLift*	0.341	**1.244**	**0.364**	1.687	0.365	**0.896**
SD XL + *CompLift*	**0.342**	1.216	**0.364**	**1.706**	**0.367**	0.890

quality of generated images.

Ablation Study: Correlation of *CompLift* with CLIP

Additionally, we show the correlation of *CompLift* with CLIP. In Figure 5.16 and 5.17, we provide examples of the correlation between CLIP and *CompLift* for text-to-image generation. The figures show the generated images for 3 prompts involving *a black car and a white clock*, *a turtle with a bow* and *a frog and a mouse*. For these prompts, we find that the diffusion models would typically ignore the clock, bow, and mouse components, leading to low CLIP scores for many generated images. Meanwhile, we observe a strong correlation between the CLIP and *CompLift* scores for these prompts, indicating the effectiveness of *CompLift* for complex prompts. For the images with low CLIP scores, they typically have lower number of activated pixels using *CompLift* for these ignored objects.

One note is that for the *a turtle with a bow* prompt, we observe that most of the images would fail to generate the bow component. This explains why many data samples have low CLIP scores on the left part of Figure 5.17. However, the *CompLift* algorithm can effectively capture

the bow component, and reject those samples with low CLIP score.

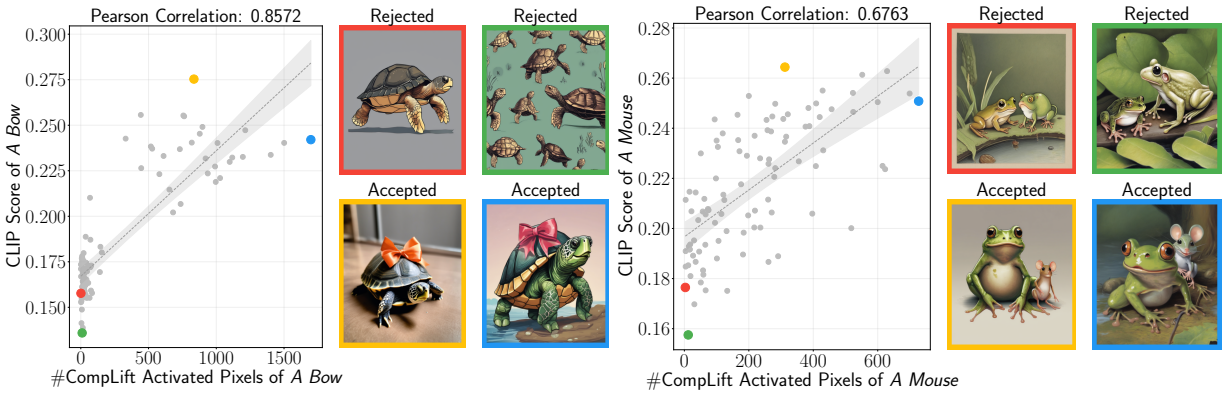


Figure 5.17: **More examples of correlation between CLIP and *CompLift*.** Prompt to generate left images: a turtle with a bow. Prompt to generate right images: a frog and a mouse.

5.7 Conclusion

We propose a novel rejection criterion *CompLift* for compositional tasks, which is based on the difference of the ELBO between the generated samples with and without the given conditions. We show that the *CompLift* criterion is effective on a 2D synthetic dataset, CLEVR Position dataset, and text-to-image compositional task. We also demonstrate that the *CompLift* criterion can be further optimized with caching strategies. In the future, we plan to explore the potential of the *CompLift* criterion on more complex compositional tasks, such as video generation and music generation.

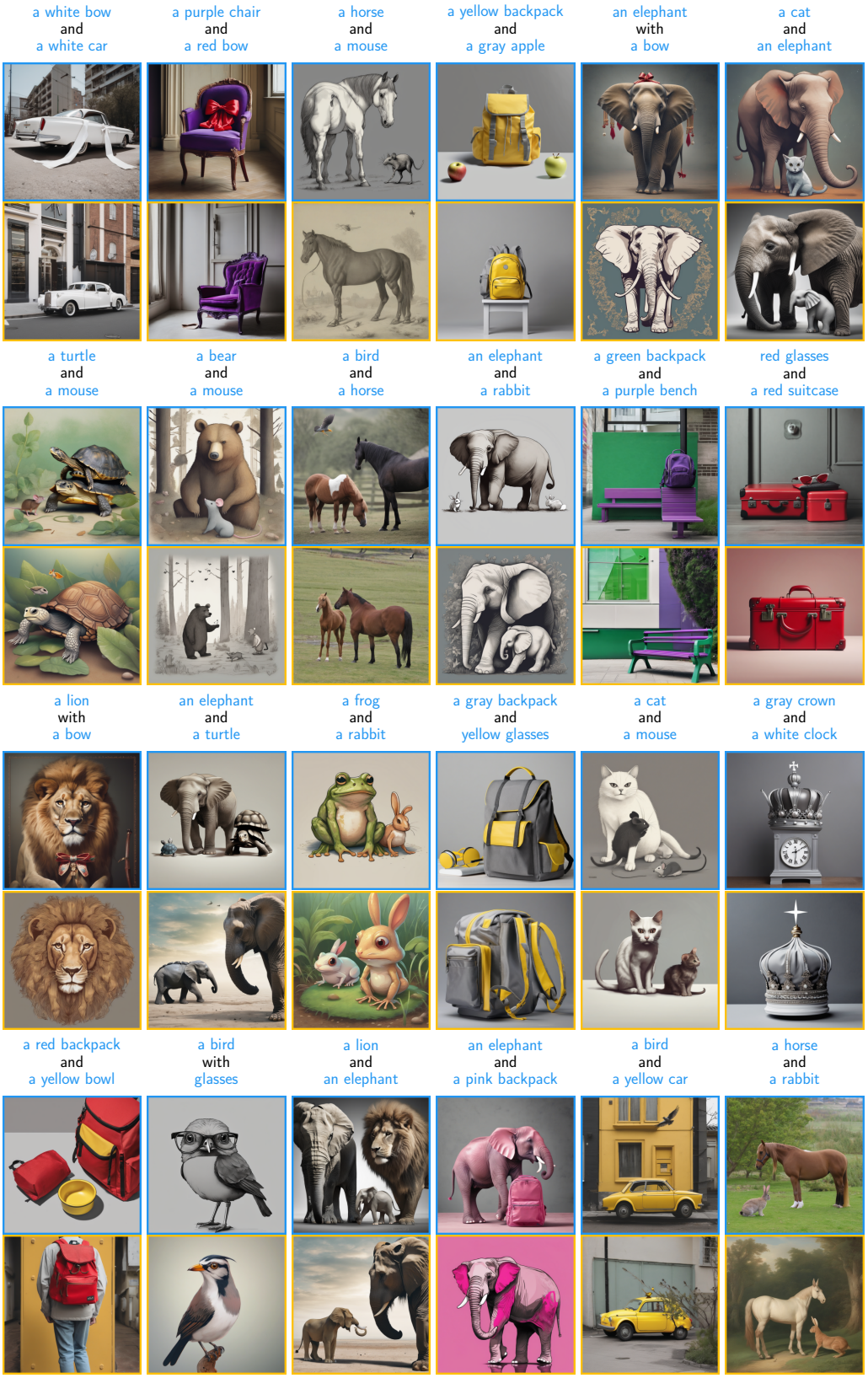


Figure 5.18: **More examples of accepted (blue) and rejected (yellow) images using SDXL + *CompLift* for text-to-image generation.**

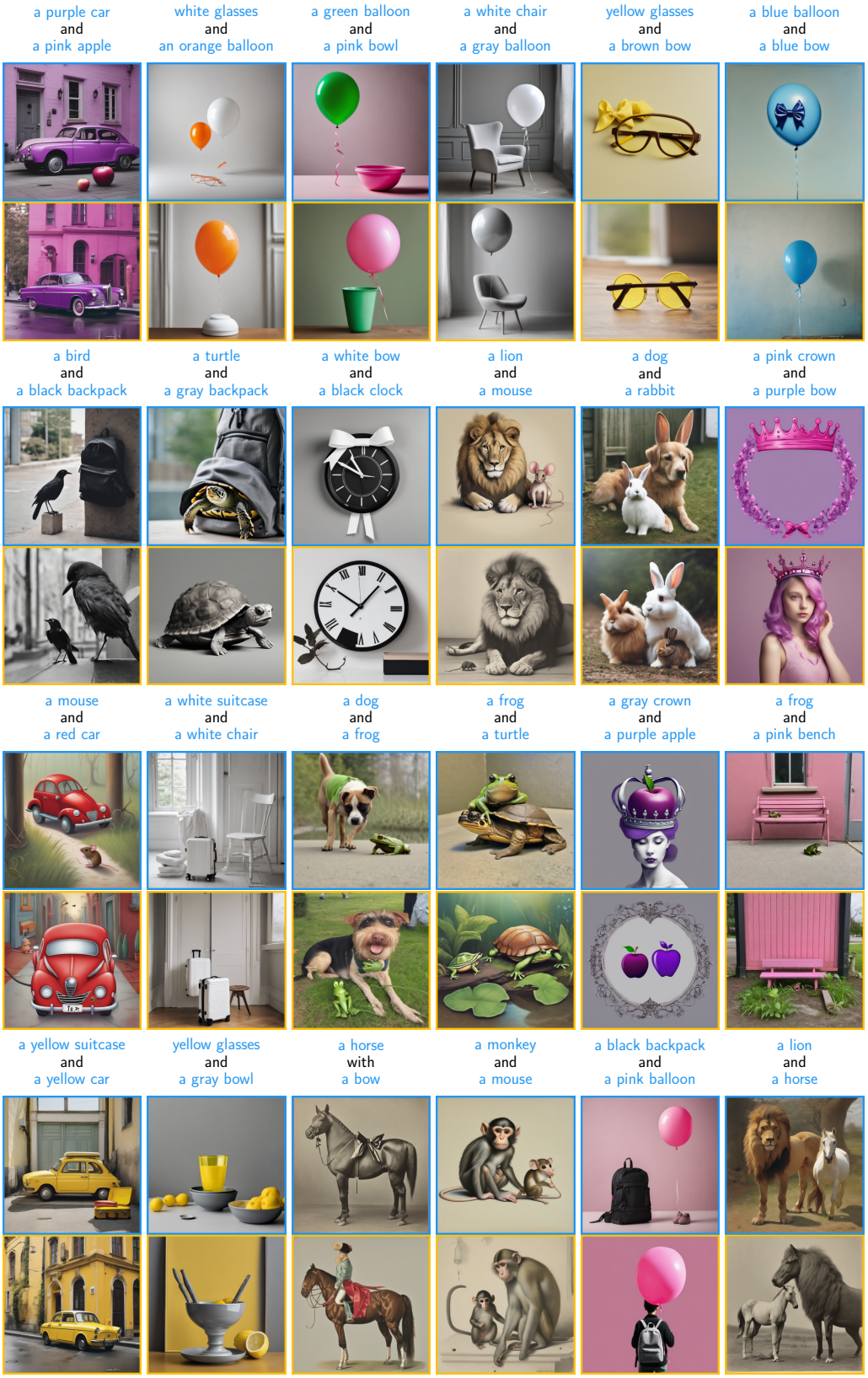


Figure 5.19: **More examples of accepted (blue) and rejected (yellow) images using SDXL + *CompLift* for text-to-image generation.**

5.8 Impact Statement

We introduce *CompLift*, a novel resampling criterion leveraging lift scores to advance the compositional capabilities of diffusion models. While we mainly focus on efficient approximation of lift scores without additional training and improving compositional generation, we acknowledge several broader implications, similar to other works on diffusion models and image generation.

As we demonstrate in this work, *CompLift* could benefit image generation applications through more reliable generation of complex, multi-attribute outputs. On the positive side, *CompLift* could enhance creative tools through more precise and reliable image generation, potentially enabling more sophisticated applications in art, design, and entertainment. However, like other works on diffusion models, *CompLift* could potentially be misused to create misleading or deepfake contents, if not properly managed [117]. These contents could be more consistent with the intention of the user, leading to potential societal harms such as scams and misinformation. Additionally, biases in training data could be amplified through the resampling process, potentially perpetuating societal biases and underrepresentation in the generated images. We encourage work building on our approach to evaluate not only technical performance but also the risks and ethical implications of the approach.

5.9 Acknowledgments

Chapter 5, in part, has been submitted for publication of the material as it may appear in Chenning Yu, Sicun Gao, "Improving Compositional Generation with Diffusion Models Using Lift Scores", ICML, 2025. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion and Future Directions

In this disseration, we present a unified framework for compositional value-based methods for both planning and generative modeling. otivated by the challenge of generalizing from simple training settings to complex, high-dimensional, or relationally structured tasks, we proposed a family of methods that construct value functions or scoring mechanisms with inherent compositionality. These methods enable generalization through logical or graph-based abstractions, providing an alternative to monolithic training on large datasets.

We demonstrate the advantage of this framework with 4 applications, i.e., sampling-based motion planning, multi-agent pathfinding, multi-agent safety enforcement, and compositional generative modeling. In sampling-based planning, our GNN-based path explorer and smoother significantly reduced the computational cost of collision checking on arbitrarily sampled graphs. In multi-agent pathfinding, our Graph Transformer heuristic generalized to team sizes multiple times larger than those seen during training, while maintaining bounded suboptimality. In safe multi-agent navigation, control admissibility models learned through GNNs enforced safety constraints across novel configurations and large agent populations. Finally, in the generative modeling domain, our lift-score-based rejection sampling technique improved controllability in diffusion models, particularly for rare and structured combinations of object constraints.

The shared principle among these applications is to leverage compositional structure in problem domains. Rather than relying solely on increased data or model capacity, we introduce inductive biases into the learning process via graph structures, attention mechanisms, and logical decompositions for better generalization and data efficiency. This approach also unlocks exciting possibilities for zero-shot transfer, scalability to larger systems, and integration with classical data-driven routines.

Future Directions

We would like to point out several promising directions for future research:

1. **Understand When Compositionality is (Not) Useful.** While we show the compositional framework is useful across various applications, it is important to understand the limitations and scenarios where compositionality may not be beneficial. Investigating the types of tasks or environments where compositional structures lead to improved performance versus those where they do not could provide insights into the underlying principles of generalization in AI systems. This understanding could inform future research on when to apply compositional methods and when alternative approaches may be more effective.

2. **Find How to Compose Automatically.** One limitation of the current framework is that the compositional structures are often hand-engineered or designed based on domain knowledge. With foundation models like Large Language Models and LLM-based agents, there is potential to find these decomposable structures and the basic compositional component directly given the description of the task. This could lead to more adaptive and flexible compositional systems that can automatically discover useful abstractions.

3. **Continue Learning During Deployment.** When deploying an already-learned compositional framework during inference, it is possible that the online data is also helpful to

improve the performance of the value function component. Developing methods for online or continual learning that can incorporate new data during inference could enhance the adaptability and robustness of value functions.

In summary, we present a unified and data-efficient framework that brings compositional value-based functions into learning-based planning and generation. By embedding inductive structures into value functions and leveraging the modularity of graph and logic-based abstractions, our approach bridges the gap between scalable deep learning and structured decision-making. The proposed methods demonstrate robust generalization, improved controllability, and adaptability across a wide range of domains. We hope this study serves as a foundation for future research in compositional intelligence, making AI systems that can reason, plan, and generate in a principled and scalable way.

Bibliography

[1] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, Naples, Italy, June 2019.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[3] Senthil Hariharan Arul, Adarsh Jagan Sathyamoorthy, Shivang Patel, Michael W. Otte, Huan Xu, Ming C. Lin, and Dinesh Manocha. Lswarm: Efficient collision avoidance for large swarms with coverage constraints in complex urban scenes. *IEEE Robotics Autom. Lett.*, 4(4):3940–3947, 2019.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y Hadaegh. Probabilistic and distributed control of a large-scale swarm of autonomous agents. *IEEE Transactions on Robotics*, 33(5):1103–1123, 2017.

[6] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters*, 2(4):1941–1947, 2017.

[7] Zhipeng Bao, Yijun Li, Krishna Kumar Singh, Yu-Xiong Wang, and Martial Hebert. Separate-and-enhance: Compositional finetuning for text-to-image diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–10, 2024.

[8] José Barbosa, Paulo Leitão, Emmanuel Adam, and Damien Trentesaux. Dynamic self-organization in holonic multi-agent manufacturing systems: The adacor evolution. *Computers in industry*, 66:99–111, 2015.

[9] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *European Conference on Artificial Intelligence*, pages 961–962, 2014.

[10] Jerome Barraquand, Bruno Langlois, and J-C Latombe. Numerical potential field techniques for robot path planning. *IEEE transactions on systems, man, and cybernetics*, 22(2):224–241, 1992.

[11] Jérôme Barraquand and Jean-Claude Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993.

[12] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4502–4510, 2016.

[13] Mayur J. Bency, Ahmed Hussain Qureshi, and Michael C. Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 3965–3972. IEEE, 2019.

[14] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[15] Mohak Bhardwaj, Sanjiban Choudhury, Byron Boots, and Siddhartha S. Srinivasa. Leveraging experience in lazy search. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019.

[16] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy PRM. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, April 24-28, 2000, San Francisco, CA, USA*, pages 521–528. IEEE, 2000.

[17] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[18] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *CoRR*, abs/2103.03012, 2021.

[19] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 255–264, 1997.

[20] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2012.

[21] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Seleckỳ. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849, 2015.

[22] J. Chase Kew, Brian Ichter, Maryam Bandari, Tsang-Wei Edward Lee, and Aleksandra Faust. Neural collision clearance estimator for batched motion planning. In Steven M. LaValle, Ming Lin, Timo Ojala, Dylan Shell, and Jingjin Yu, editors, *Algorithmic Foundations of Robotics XIV*, pages 73–89, Cham, 2021. Springer International Publishing.

[23] Bernard Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13(3):488–507, 1984.

[24] Hila Chefer, Yuval Alaluf, Yael Vinker, Lior Wolf, and Daniel Cohen-Or. Attend-and-excite: Attention-based semantic guidance for text-to-image diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4):1–10, 2023.

[25] Binghong Chen, Bo Dai, Qinjie Lin, Guo Ye, Han Liu, and Le Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[26] Huanran Chen, Yinpeng Dong, Zhengyi Wang, Xiao Yang, Chengqi Duan, Hang Su, and Jun Zhu. Robust classification via a single diffusion model. *arXiv preprint arXiv:2305.15241*, 2023.

[27] Minghao Chen, Iro Laina, and Andrea Vedaldi. Training-free layout control with cross-attention guidance. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5343–5353, 2024.

[28] Moëz Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 317–322. IEEE Robotics and Automation Society, 1999.

[29] Howie M Choset, Kevin M Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[30] Benjamin J. Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In Dieter Fox, Lydia E. Kavraki, and Hanna Kurniawati, editors, *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014.

[31] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[32] Hanjun Dai, Yujia Li, Chenglong Wang, Rishabh Singh, Po-Sen Huang, and Pushmeet Kohli. Learning transferable graph exploration. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2514–2525, 2019.

[33] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[34] Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[36] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[37] Rosen Diankov and James Kuffner. Randomized statistical path planning. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*, pages 1–6. IEEE, 2007.

[38] Bruce Randall Donald, Patrick G. Xavier, John F. Canny, and John H. Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993.

[39] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[40] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P. Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In M. Arif Wani, Feng Luo, Xiaolin Andy Li, Dejing Dou, and Francesco Bonchi, editors, *19th IEEE International Conference on Machine Learning and Applications, ICMLA 2020, Miami, FL, USA, December 14-17, 2020*, pages 19–24. IEEE, 2020.

[41] Yilun Du, Conor Durkan, Robin Strudel, Joshua B Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Sussman Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc. In *International conference on machine learning*, pages 8489–8510. PMLR, 2023.

[42] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[43] Scott Emmons, Ajay Jain, Michael Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[44] John Enright and Peter R Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots*, pages 33–38, 2011.

[45] Ben Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15220–15231, 2019.

[46] Ariel Felner, Meir Goldenberg, Guni Sharon, Roni Stern, Tal Beja, Nathan R. Sturtevant, Jonathan Schaeffer, and Robert Holte. Partial-expansion a* with selective node generation. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.

[47] Ariel Felner, Jiaoyang Li, Eli Boyarski, Hang Ma, Liron Cohen, TK Satish Kumar, and Sven Koenig. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 83–87, 2018.

[48] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. *arXiv preprint arXiv:2212.05032*, 2022.

[49] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. Layoutgpt: Compositional visual planning and generation with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[50] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[51] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *CoRR*, abs/1405.5848, 2014.

[52] Shuzhi Sam Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002.

[53] Malik Ghallab and Dennis G Allard. A: An efficient near admissible heuristic search algorithm. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 789–791. Citeseer, 1983.

[54] Edward N Gilbert. Random plane networks. *Journal of the society for industrial and applied mathematics*, 9(4):533–543, 1961.

[55] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[56] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25, 2006.

[57] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J. Russell, and Anca D. Dragan. Inverse reward design. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6765–6774, 2017.

[58] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The provable virtue of laziness in motion planning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6161–6165. ijcai.org, 2019.

[59] Dongkun Han, Lixing Huang, and Dimitra Panagou. Approximating the region of multi-task coordination via the optimal lyapunov-like barrier function. In *2018 Annual American Control Conference, ACC 2018, Milwaukee, WI, USA, June 27-29, 2018*, pages 5070–5075. IEEE, 2018.

[60] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[62] Eric Heiden, Luigi Palmieri, Sven Koenig, Kai Oliver Arras, and Gaurav S. Sukhatme. Gradient-informed path smoothing for wheeled mobile robots. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1710–1717. IEEE, 2018.

[63] Eduardo G Hernández-Martínez, Eduardo Aranda-Bricaire, and F Alkhateeb. *Convergence and collision avoidance in formation control: A survey of the artificial potential functions approach*. INTECH Open Access Publisher Rijeka, Croatia, 2011.

[64] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[65] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[66] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[67] Binxuan Huang and Kathleen M Carley. Residual or gate? towards deeper graph neural networks for inductive graph representation learning. *arXiv preprint arXiv:1904.08035*, 2019.

[68] Kaiyi Huang, Kaiyue Sun, Enze Xie, Zhenguo Li, and Xihui Liu. T2i-compbench: A comprehensive benchmark for open-world compositional text-to-image generation. *Advances in Neural Information Processing Systems*, 36:78723–78747, 2023.

[69] Taoan Huang, Bistra Dilkina, and Sven Koenig. Learning node-selection strategies in bounded suboptimal conflict-based search for multi-agent path finding. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2021.

[70] Taoan Huang, Sven Koenig, and Bistra Dilkina. Learning to resolve conflicts for multi-agent path finding with conflict-based search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11246–11253, 2021.

[71] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7087–7094. IEEE, 2018.

[72] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics Autom. Lett.*, 4(3):2407–2414, 2019.

[73] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[74] Lucas Janson and Marco Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions - extended version. *CoRR*, abs/1306.3532, 2013.

[75] P Jiménez, Federico Thomas, and Carme Torras. Collision detection algorithms for motion planning. In *Robot motion planning and control*, pages 305–343. Springer, 1998.

[76] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

[77] Tom Jurgenson and Aviv Tamar. Harnessing reinforcement learning for neural motion planning. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019.

[78] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[79] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Res.*, 30(7):846–894, 2011.

[80] Shyamgopal Karthik, Karsten Roth, Massimiliano Mancini, and Zeynep Akata. If at first you don't succeed, try, try again: Faithful diffusion-based text-to-image generation by selection. *arXiv preprint arXiv:2305.13308*, 2023.

[81] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017.

[82] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.

[83] Arbaaz Khan, Alejandro Ribeiro, Vijay Kumar, and Anthony G. Francis. Graph neural networks for motion planning. *CoRR*, abs/2006.06248, 2020.

[84] Arbaaz Khan, Ekaterina Tolstaya, Alejandro Ribeiro, and Vijay Kumar. Graph policy gradients for large scale robot control. In *Conference on robot learning*, pages 823–834. PMLR, 2020.

[85] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[86] Dongjun Kim, Yeongmin Kim, Se Jung Kwon, Wanmo Kang, and Il-Chul Moon. Refining generative process with discriminator guidance in score-based diffusion models. *arXiv preprint arXiv:2211.17091*, 2022.

[87] Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *arXiv preprint arXiv:2207.02505*, 2022.

[88] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[89] Koichi Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Trans. Robotics Autom.*, 7(3):267–277, 1991.

[90] Xianghao Kong, Ollie Liu, Han Li, Dani Yogatama, and Greg Ver Steeg. Interpretable diffusion via information decomposition. *arXiv preprint arXiv:2310.07972*, 2023.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[92] Vijay Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. In *Robotics Research*, pages 41–58. Springer, 2017.

[93] Edward Lam and Pierre Le Bodic. New valid inequalities in branch-and-cut-and-price for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 184–192, 2020.

[94] Edward Lam, Pierre Le Bodic, Daniel Harabor, and Peter J Stuckey. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144:105809, 2022.

[95] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[96] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 473–479. IEEE Robotics and Automation Society, 1999.

[97] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric P. Xing, and Ruslan Salakhutdinov. Gated path planning networks. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2953–2961. PMLR, 2018.

[98] Alexander C Li, Mihir Prabhudesai, Shivam Duggal, Ellis Brown, and Deepak Pathak. Your diffusion model is secretly a zero-shot classifier. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2206–2217, 2023.

[99] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, volume 2019, pages 442–449, 2019.

[100] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12353–12362, 2021.

[101] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11785–11792, 2020.

[102] Qingbiao Li, Weizhe Lin, Zhe Liu, and Amanda Prorok. Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics and Automation Letters*, 6(3):5533–5540, 2021.

[103] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[104] Iou-Jen Liu, Raymond A Yeh, and Alexander G Schwing. Pic: permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, pages 590–602. PMLR, 2020.

[105] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. In *European Conference on Computer Vision*, pages 423–439. Springer, 2022.

[106] Yujian Liu, Yang Zhang, Tommi Jaakkola, and Shiyu Chang. Correcting diffusion generation through resampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8713–8723, 2024.

[107] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6379–6390, 2017.

[108] Ryan J Luna and Kostas E Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[109] Kevin M. Lynch and Matthew T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robotics Res.*, 15(6):533–556, 1996.

[110] Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7643–7650, 2019.

[111] Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, et al. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025.

[112] Ratnesh Madaan, Sam Zeng, Brian Okorn, and Sebastian Scherer. Learning adaptive sampling distributions for motion planning by self-imitation. In *Workshop on Machine Learning in Robot Motion Planning, IEEE IROS*, 2018.

[113] Aditya Mandalika, Sanjiban Choudhury, Oren Salzman, and Siddhartha S. Srinivasa. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 745–753. AAAI Press, 2019.

[114] Aditya Mandalika, Oren Salzman, and Siddhartha S. Srinivasa. Lazy receding horizon a* for efficient path planning in graphs with expensive-to-evaluate edges. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, pages 476–484. AAAI Press, 2018.

[115] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[116] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[117] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *ACM computing surveys (CSUR)*, 54(1):1–41, 2021.

[118] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

[119] Jörg P Müller and Klaus Fischer. Application impact of multi-agent systems and technologies: A survey. In *Agent-oriented software engineering*, pages 27–53. Springer, 2014.

[120] Byeonghu Na, Yeongmin Kim, Minsang Park, Donghyeok Shin, Wanmo Kang, and Il-Chul Moon. Diffusion rejection sampling. *arXiv preprint arXiv:2405.17880*, 2024.

[121] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[122] Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa C. Smith, and Jelena Kovacevic. Generalized value iteration networks: Life beyond lattices. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6246–6253. AAAI Press, 2018.

[123] Keisuke Okumura, Ryo Yonetani, Mai Nishimura, and Asako Kanezaki. Ctrms: Learning to construct cooperative timed roadmaps for multi-agent path planning in continuous spaces. *arXiv preprint arXiv:2201.09467*, 2022.

[124] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[125] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[126] Judea Pearl and Jin H Kim. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):392–399, 1982.

[127] Sven Mikael Persson and Inna Sharf. Sampling-based a* algorithm for robot path-planning. *Int. J. Robotics Res.*, 33(13):1683–1708, 2014.

[128] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[129] Amanda Prorok and Vijay Kumar. Privacy-preserving vehicle assignment for mobility-on-demand systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1869–1876, 2017.

[130] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017.

[131] Zipeng Qi, Lichen Bai, Haoyi Xiong, and Zeke Xie. Not all noises are created equally: Diffusion noise selection and optimization. *arXiv preprint arXiv:2407.14041*, 2024.

[132] Zengyi Qin, Kaiqing Zhang, Yuxiao Chen, Jingkai Chen, and Chuchu Fan. Learning Safe Multi-Agent Control with Decentralized Neural Barrier Certificates. In *Conference on Learning Representations*. Conference on Learning Representations, jan 2021.

[133] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37(1):48–66, 2021.

[134] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[135] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[136] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[137] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[138] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427, 1979.

[139] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In Jeff A. Bilmes and Andrew Y. Ng, editors, *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461. AUAI Press, 2009.

[140] Benjamin Riviere, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, 5(3):4249–4256, 2020.

[141] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[142] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[143] Qandeel Sajid, Ryan Luna, and Kostas Bekris. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *International symposium on combinatorial search*, volume 3, 2012.

[144] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *CVPR*, 2020.

[145] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[146] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[147] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[148] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in Neural Information Processing Systems*, 35:25278–25294, 2022.

[149] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[150] Dhruv Shah, Benjamin Eysenbach, Nicholas Rhinehart, and Sergey Levine. RECON: rapid exploration for open-world navigation with latent goal models. *CoRR*, abs/2104.05859, 2021.

[151] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[152] Guanya Shi, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. Neural-swarm: Decentralized close-proximity multirotor control using learned interactions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3241–3247. IEEE, 2020.

[153] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[154] Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. *arXiv preprint arXiv:1804.00846*, 2018.

[155] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[156] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[157] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[158] Marlin P. Strub and Jonathan D. Gammell. Advanced bit* (abit*): Sampling-based planning with advanced graph-search techniques. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 130–136. IEEE, 2020.

[159] Robin A. M. Strudel, Ricardo Garcia, Justin Carpentier, Jean-Paul Laumond, Ivan Laptev, and Cordelia Schmid. Learning obstacle representations for neural motion planning. *CoRR*, abs/2008.11174, 2020.

[160] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[161] Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. Value iteration networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2146–2154, 2016.

[162] Hao Tang, Zhiao Huang, Jiayuan Gu, Bao-Liang Lu, and Hao Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15811–15822. Curran Associates, Inc., 2020.

[163] Ekaterina Tolstaya, James Paulos, Vijay Kumar, and Alejandro Ribeiro. Multi-robot coverage and exploration using spatial graph neural networks. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8944–8950. IEEE, 2020.

[164] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[165] J. P. van den Berg and M. H. Overmars. Prioritized motion planning for multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 430–435, 2005.

[166] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[167] Jur P Van Den Berg and Mark H Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[168] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[169] Petar Velickovic, Lars Buesing, Matthew C. Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[170] Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[171] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, pages 3260–3267. IEEE, 2011.

[172] Shuo Wan, Jiaxun Lu, and Pingyi Fan. Semi-centralized control for multi robot formation. In *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, pages 31–36. IEEE, 2017.

[173] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.

[174] Ruichen Wang, Zekang Chen, Chen Chen, Jian Ma, Haonan Lu, and Xiaodong Lin. Compositional text-to-image synthesis with attention map control of diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 5544–5552, 2024.

[175] Qiucheng Wu, Yujian Liu, Handong Zhao, Trung Bui, Zhe Lin, Yang Zhang, and Shiyu Chang. Harnessing the spatial-temporal attention of diffusion models for high-fidelity text-to-image synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7766–7776, 2023.

[176] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[177] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[178] Yilun Xu, Mingyang Deng, Xiang Cheng, Yonglong Tian, Ziming Liu, and Tommi Jaakkola. Restart sampling for improving generative processes. *Advances in Neural Information Processing Systems*, 36:76806–76838, 2023.

[179] Feng Xue and Panganamala R Kumar. The number of neighbors needed for connectivity of wireless networks. *Wireless networks*, 10(2):169–181, 2004.

[180] Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Hashemi. Neural execution engines: Learning to execute subroutines. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[181] Ge Yang, Amy Zhang, Ari S. Morcos, Joelle Pineau, Pieter Abbeel, and Roberto Calandra. Plan2vec: Unsupervised representation learning by latent plans. In Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020*, volume 120 of *Proceedings of Machine Learning Research*, pages 935–946. PMLR, 2020.

[182] Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. In *Proceedings of the 35rd International Conference on Neural Information Processing Systems*, 2021.

[183] Chenning Yu, Hongzhan Yu, and Sicun Gao. Learning control admissibility models with graph neural networks for multi-agent navigation, 2022.

[184] Jingjin Yu and Steven M LaValle. Planning optimal paths for multiple robots on graphs. In *2013 IEEE International Conference on Robotics and Automation*, pages 3612–3617. IEEE, 2013.

[185] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[186] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[187] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020.

[188] Clark Zhang, Jinwook Huh, and Daniel D. Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3654–3661. IEEE, 2018.

[189] Ruipeng Zhang, Chenning Yu, Jingkai Chen, Chuchu Fan, and Sicun Gao. Learning-based motion planning in dynamic environments using gnns and temporal encoding. *CoRR*, abs/2210.08408, 2022.

[190] Ray Y Zhong, Xun Xu, Eberhard Klotz, and Stephen T Newman. Intelligent manufacturing in the context of industry 4.0: a review. *Engineering*, 3(5):616–630, 2017.

[191] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[192] Lifeng Zhou, Vishnu D Sharma, Qingbiao Li, Amanda Prorok, Alejandro Ribeiro, and Vijay Kumar. Graph neural networks for decentralized multi-robot submodular action selection. *arXiv preprint arXiv:2105.08601*, 2021.

[193] Zhijie Zhu, Edward Schmerling, and Marco Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 54th IEEE conference on decision and control (CDC)*, pages 835–842. IEEE, 2015.

[194] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.