Distributed Dexterous Manipulation with Spatially Conditioned Multi-Agent Transformers

Sarvesh Patil
sarveshp@andrew.cmu.edu
Carnegie Mellon University, The Robotics Institute
Pittsburgh, PA, United States

Abstract: Distributed Dexterous Manipulation (DDM) is a novel paradigm that presents significant control challenges due to high action-space redundancy, inter-robot cooperation, and dynamic object-robot interactions. This paper introduces a framework based on spatially conditioned Multi-Agent Transformers (MATs) to efficiently learn robust control policies for a DDM system grounded in an array of 64 soft delta robots arranged in an 8×8 grid. Our three core contributions are: (i) an MAT with adaptive layer norm for compute efficiency, (ii) spatial contrastive embeddings to ground transformer embeddings in the spatial configuration of the robots, and (iii) an MAT-based behavior cloning method fine-tuned using Soft Actor Critic. We also propose an action selection formulation to analyze the trade-off between task performance and the number of robots utilized. Our experiments show that MATs iteratively refine their actions through the stacked attention blocks. This further informs the benefit of spatial conditioning in transformers to learn DDM policies. We demonstrate long-horizon planar manipulation tasks with objects of various geometries in simulation and real-world. Finally, we show how action selection mitigates robot maintenance by reducing wear and tear due to inter-robot collisions while maintaining the ability to manipulate objects along various trajectories in the real-world, achieving an average error of ~ 1.5 cm, while using $\sim 65\%$ fewer robots.

Keywords: Distributed Dexterous Manipulation, Multi Agent Reinforcement Learning, Soft Robots

1 Introduction

Enabling robots to dexterously manipulate a wide variety of objects remains a fundamental challenge. Distributed manipulation systems, where multiple actuators impart desired motion on an object through a combination of external forces, offer a promising approach. Examples include smart conveyors [1, 2] and linear actuator arrays [3, 4] that move objects through coordinated patterns. Recently, Patil et al. [5] introduced delta arrays, a distributed dexterous manipulation (**DDM**) system comprising an 8×8 array of 3-DoF compliant delta robots as shown in Fig. 1. This system can apply forces dynamically by engaging varying groups of robots, allowing for dexterous manipulation of diverse objects.

Learning effective control policies for delta arrays is particularly challenging due to: (1) the extreme redundancy in the collective action space (192-DoF) resulting in highly multi-modal dynamics; (2) the need for robots to communicate intent and cooperate within dynamically changing neighborhoods based on object state; and (3) the requirement for sample-efficient learning methods suitable for complex, contact-rich interactions. However, controlling such **DDM** systems presents significant hurdles. Prior works have focused on rotary actuator arrays [6, 1] or linear actuators [7], limiting their applicability to the full 3-DoF capabilities needed for dexterous tasks.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

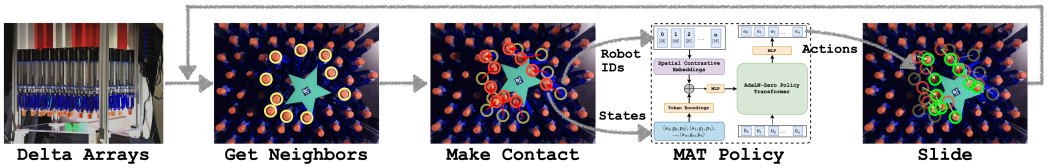


Figure 1: We present Distributed Dexterous Manipulation as a novel paradigm for cooperative manipulation using arrays of soft delta robots. We propose a spatially grounded Multi-Agent Transformer to train sample-efficient policies using off-policy reinforcement learning and behavior cloning.

To address these challenges, we propose a learning framework centered around spatially conditioned Multi-Agent Transformers (MATs) for manipulating objects on an $\mathbb{XY}$ plane. Transformers [8] are well-suited for **DDM** due to their ability to handle robot sequences of varying lengths (dynamic neighborhoods) and learn effective representations between objects and robots. To explicitly ground the attention layers in MATs to the physical layout of delta arrays, we introduce Spatial Contrastive Embeddings (SCEs), which encode the spatial arrangement of the robots in the array. Further, to address sample efficiency, we employ a two-phase learning strategy: (i) pre-training the MAT policy via Behavior Cloning (*MATBC*) using demonstrations from a visual servoing expert, (ii) fine-tuning with off-policy Reinforcement Learning using Soft Actor-Critic (*MATSAC*) [9]. We refer to this two-phase fine-tuned policy as *MATBC-FT*. Unlike the hand-engineered visual servoing policy, which follows fixed flow vectors, *MATBC-FT* learns to anticipate object dynamics and adaptively coordinates robots, resulting in smoother trajectories and more robust manipulation across various object geometries. Further, we experiment with reward shaping to affect the $\mathbb{Z}$ component of the action space to select or deselect robots. This induces the policy to learn an optimal trade-off between accurately manipulating objects and selecting the number of agents. This also mitigates physical wear and prevents entanglement of the compliant delta linkages on the real hardware.

The core contributions of this paper are:

1. A framework using Spatially Conditioned Multi-Agent Transformers combined with Behavior Cloning and Reinforcement Learning Fine-tuning (*MATBC-FT*) for learning sample-efficient and robust policies for complex **DDM** tasks
2. Spatial Contrastive Embeddings (SCEs) as a novel method to explicitly encode spatial relationships between robots, enhancing coordination within the MAT for **DDM**.
3. An analysis of reward shaping mechanisms for efficient robot utilization by learning a trade-off between task success and the number of active agents.
4. Providing empirical validation and analysis on the benefits of iterative refinement from message-passing through transformer attention layers for cooperative manipulation tasks.

2 Related Work

Distributed Manipulation: Distributed manipulation systems employ multiple actuators, often in 2D tessellating structures, for coordinated object motions. Implementations range from air-jet arrays [10] to surfaces with actuated joints [11, 1, 2] and actuated workbenches [12, 13]. While some works have explored 3D cooperative pushing [14, 15, 16], these systems are limited by the end-effector dexterity and workspace constraints [7]. The delta arrays system [5], used in our work, represents a step towards **DDM** by offering a dense array of 3-DoF compliant delta robots. They enable complex interactions but also introduce significant control challenges due to the high dimensionality of their action spaces and the need for coordination within the shared workspace of the robots.

Behavior Cloning (BC) and Reinforcement Learning (RL) for Dexterous Manipulation: BC learns policies by imitating expert demonstrations [17, 18, 19, 20], while RL develops strategies through trial-and-error [21]. RL has shown success in complex single-agent dexterous manipulation [22, 23, 24] using both on-policy (e.g., Proximal Policy Optimization (PPO) [25]) and off-policy

(e.g., Twin Delayed Deep Deterministic Policy Gradient (TD3) [26] Soft Actor-Critic (SAC) [9]) methods. However, RL often suffers from poor sample efficiency, requiring extensive interaction [27], which is problematic for real robots. Combining BC pre-training with RL fine-tuning has been proposed in literature to learn robust control policies [28, 29, 30, 31]. Offline RL has also been explored for multi-agent settings [32, 33], but applying these methods effectively to the fine-grained manipulation tasks in **DDM** remains unexplored. Our methods utilize the sequence modeling capacity of transformers to learn **DDM** policies.

Multi-Agent RL (MARL) for Dexterous Manipulation: DDM inherently involves multiple cooperating robots. MARL often employs the paradigm of Centralized Training with Decentralized Execution. Methods used in Jakob et al. [34], Rashid et al. [35], Son et al. [36] learn global value functions and decompose into local policies. Effective credit assignment becomes crucial for these methods. Transformers, on the other hand, have been proposed [37, 38] as a Centralized Training, Centralized Execution method. They have shown great ability to learn sequential representations and handle varying input lengths [39, 19, 40]. Wen et al. [38] proposed an MAT with PPO for simulated bi-manual tasks in simulation. Our methods induce implicit spatial reasoning needed for multi-robot systems like the delta arrays and show zero-shot transfer from simulation to real-world.

3 Problem Formulation and Preliminaries

3.1 Delta Arrays for Distributed Dexterous Manipulation

We perform our real-world evaluations on the delta arrays introduced by Patil et al. [5]. The array consists of 64 compliant delta robots, i.e., fingers, [41] inspired by the original delta mechanism [42]. Each delta finger has three linear actuators, enabling control in the $\mathbb{XY}$ axes within an oval-shaped workspace of ~ 2.5 cm radius, and 10 cm for the $\mathbb{Z}$ axis. The result is a 192 DoF combined action space across the array. The delta robots are densely arranged in a hexagonal grid, with centers spaced ~ 4.35 cm apart. This overlap of individual workspaces allows subsets of robots to collaborate on common tasks as well as large objects. We observe states using a bottom-up camera positioned beneath a glass plane that originates the world frame. The robot positions and workspaces are visualized in Fig. 6. To accelerate training and testing, we implement a simulation environment in MuJoCo. Each delta robot is modeled as a floating capsule 3 DoF fingertip with real-world workspace constraints.

3.2 Problem Definition

We formulate the dexterous distributed manipulation problem as a Markov Decision Process (MDP) given by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S} = \prod_i \mathcal{S}^i$ is the state space and $\mathcal{S}^i$ represents the individual state space of robot i, and $\mathcal{A} = \prod_i \mathcal{A}^i$ is the action space and each $\mathcal{A}^i$ represents the individual action space of robot $i \in \{1, 2, ...64\}$. At each timestep t, we randomly generate an initial pose and a goal pose for an object being manipulated. We use an image processing pipeline to obtain pairs of robots and object-boundary points using nearest neighbor search, and eliminate robots directly on top of the objects and whose workspaces do not overlap with the object boundary as shown in Appendix 9.1. We define this neighborhood of robots with their indices sampled as a set $\mathcal{N}_t \subseteq \{1, 2, ...64\}$. Each robot's state contains a set of initial boundary points s_t^{INIT}, the position of the robot s_t^{PROP}, and the set of goal boundary points s_t^{GOAL}, resulting in a robot state space of $\mathcal{S}^i \in \mathbb{R}^{2+3+2}$. The actions taken by the robots are described as a set of 3D displacement vectors $a_t = (a_x, a_y, a_z) \in \mathbb{R}^3$. For planar manipulation tasks, we consider the 2D component of the action space $a_{t_{\text{xy}}} = (a_x, a_y) \in \mathbb{R}^2$, while $a_z \in \{a_z^{\text{LOW}}, a_z^{\text{HIGH}}\}$ is used as an action selection mechanism that determines $\mathcal{N}_{\text{SEL}} \subseteq \mathcal{N}_t$ with $a_z = a_z^{\text{LOW}}$. The dynamics of the environment are captured by the transition function $\mathcal{T}(s_{t+1}|s_t, a_t) \in \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and the rewards are captured by the function $r(s_t, a_t) \in \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Finally, our objective is to learn policies that maximize the expected cumulative reward: $J(\pi) = \mathbb{E}_{s_t, a_t \sim} [\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$

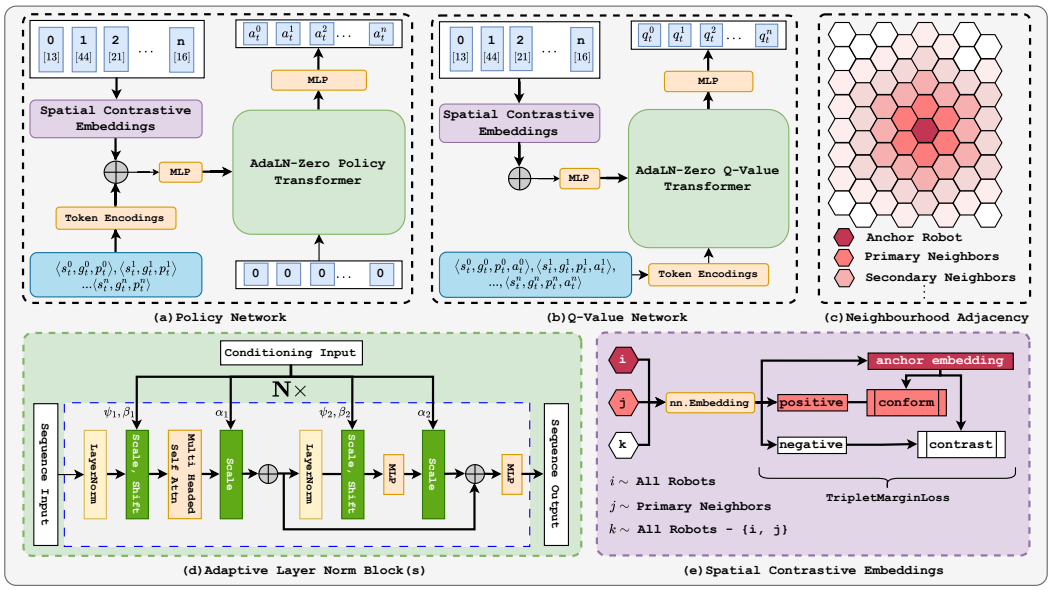


Figure 2: (a) Decoder-only MAT policy network. (b) Decoder-only MAT Q-value network. (c) Neighbourhood adjacency to select positive embeddings as primary neighbors and negative embeddings as farther away neighbors. (d) Adaptive Layer Norm Block. (e) Spatial Contrastive Embeddings using Triplet Margin Loss to create pairwise conformity and contrast in the embedding space.

4 Methods

4.1 Multi-Agent Transformers

The core of our policy representation is a Multi-Agent Transformer (MAT) [38]. Specifically, we use a *decoder-only* transformer architecture as a policy and a Q function. The policy passes the zero vector of joint actions $a_t^{0_{0:n-1}}$ to a sequence of decoder transformer blocks. Actions are iteratively conditioned on the state and position embeddings through the attention layers of the transformer (Fig. 2(a)). On the other hand, Q functions learn a joint distribution over the states and the actions to learn the expected reward distribution. Hence, we concatenate states and actions over all the robots as the sequential input to the transformer. These are conditioned on the position embeddings to obtain individual Q-values over all robots, mapping $\mathbb{E}[r] \leftarrow \mathbb{E}[Q(s_t^i, a_t^i)]$ (Fig. 2(b)).

For efficient policy learning in high-dimensional **DDM** tasks, we incorporate Adaptive Layer Normalization with Zero-Initialization (AdaLN-Zero) [43, 44, 45] as the transformer blocks. As shown in Fig. 2(d), each AdaLN-Zero layer takes the primary sequence (robot states/actions) and a conditional input to compute adaptive scaling (ψ) and shifting (β) parameters. Zero-initializing the learned scaling parameter ψ accelerates training [43]. A learned scaling parameter α is added before the residual connection for stability. ψ and β map conditional inputs to the sequential inputs, which mitigates the need to learn a complex attention map capturing representations with dynamically varying sequences of robots.

During training, we use masked Multi-Head Attention (MHA) as proposed in [46] within the transformer blocks to allow agents to implicitly coordinate by attending to relevant information from others in the sequence. Crucially, the attention mechanism is guided by Spatial Contrastive Embeddings (SCEs), which ground the representations in the local neighborhood of the delta arrays as shown in the Appendix Fig. 3. The final output of the MAT is a set of refined tokens, decoded into agent-specific actions $a_i \in \mathcal{A}$.

4.2 Spatial Contrastive Embeddings

A fundamental challenge of **DDM** systems is the stochastic ordering of the neighborhood for robots. At each timestep t neighboring robots $\mathcal{N}_t$ are arbitrarily sequenced detached from their spatial ar-

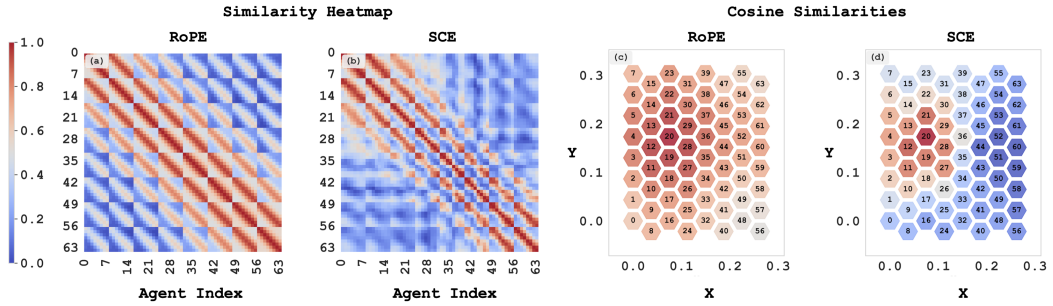


Figure 3: RoPE fails to generate embeddings that capture similarities among neighboring robots as SCE embeddings are able to do. (left) Heatmap of the cosine similarity among all the robots' embeddings. (right) Cosine similarity of an arbitrary robot (20) visualized on the delta array structure.

rangement in the real world. Standard position embeddings like Sinusoidal Position Embeddings (SPEs)[8] and Rotary Position Embeddings (RoPEs) [47] fail to capture spatial relationships in such dynamically changing agent configurations. We introduce Spatial Contrastive Embeddings (SCEs) to address this challenge. SCEs map each robot's index to a learned embedding vector $e_i \in \mathbb{R}^d$, creating a persistent spatial representation. These embeddings are trained offline using a triplet loss function:

$$\mathcal{L}_{\text{triplet}}(e_i, e_j, e_k) = \max(0, \|e_i - e_j\|_2^2 - \|e_i - e_k\|_2^2 + m) \tag{1}$$

where e_i and e_j are embeddings of physically adjacent robots, e_k are distant, and m is a margin parameter. This ensures that neighboring robots have similar embeddings while distant robots have dissimilar ones, preserving the array's spatial topology in the learned representations. We compare the 2D representations projected by RoPE and SCE in Fig. 3. SCEs serve as conditional inputs to the AdaLN-Zero layers in the MAT, enabling the transformer to reason about spatial relationships through attention even when the active robot subsets vary dynamically

4.3 Learning Phase 1: Pre-training via Behavior Cloning (*MATBC*)

To improve sample efficiency and provide a good initialization for policy learning in the highly redundant delta arrays action space, we first pre-train the MAT policy using Behavior Cloning (*MATBC*). However, providing human demonstrations for a 64-robot system is non-trivial due to the complexity of teleoperating numerous agents simultaneously. Hence, we develop a visual servoing pipeline to generate *expert* demonstrations. Following the visual servoing pipeline described in Appendix Sec. 9.1 to generate a dataset of *expert* demonstrations $\mathcal{D}_{exp}$ The *MATBC* policy, π_ϕ^{BC}, is trained by minimizing the Mean Squared Error (MSE) between its predicted actions $\mathcal{A}^\pi = \pi_\phi^{BC}(\mathcal{S})$ and the expert actions $\mathcal{A}^e$. Concurrently, we pre-train an initial critic network, Q_θ^{BC}. This critic is trained via Mean Squared Error (MSE) on a mixed dataset $\mathcal{D}_{mixed}$, containing both expert and random transitions, to predict the immediate reward r associated with a state-action pair. During BC, our Q function does not incorporate future rewards via Bellman updates due to the quasi-static nature of visual servoing data. This provides a basic value estimation before RL fine-tuning. The specific loss function is detailed in Appendix Sec. 9.2.

4.4 Learning Phase 2: Fine-tuning via Reinforcement Learning (*MATBC-FT*)

In the second phase, we train the policy and the critic using Soft Actor-Critic (SAC) [9] adapted for multi-agent transformers (*MATSAC*) as a baseline. We initialize the actor and critic networks with the pre-trained weights: $\pi_\phi \leftarrow \pi_\phi^{BC}$ and $Q_\theta \leftarrow Q_\theta^{BC}$. We refer to the resulting fine-tuned policy as *MATBC-FT*. The critic $Q_\theta(s_t, a_t)$ takes the global state $\mathcal{S}$ and joint action $\mathcal{A}$ and outputs agent-specific Q-values $[q_1, \ldots, q_N]$. To avoid overestimation bias, we train two critics $(Q_{\theta_1}, Q_{\theta_2})$ and use the minimum for target calculations. The actor π_ϕ is trained to maximize both expected future returns and entropy, facilitating exploration in the high-dimensional action space.

To address execution costs of controlling the robots, we shape reward functions with a discrete and a continuous cost over the policy's action selection. Let $\mathcal{N}_{\text{SEL}} \subseteq \mathcal{N}$ be the selected robots with $a_z = 10cm$, and $\mathcal{A}_{\text{SEL}_{\text{xy}}}$ be their corresponding 2D actions. We study four rewards to select a sparse set of robots to execute movements for each step we term as **action selection**:

1. The original reward function (OG): $r_{\text{OG}} \leftarrow \frac{1}{c\delta^2+\epsilon}$, where δ is given by $||s_t^{\text{GOAL}} - s_t^{\text{FINAL}}||_2^2$, c is a scaling factor that controls the shape of the reward function, and ϵ is a bounding factor. We set $\epsilon = 0.01$ to bound the rewards between 0 and 100
2. Discrete Execution Cost (DEC): $r_{\text{DEC}} \leftarrow r_{\text{OG}} - \lambda_1 \frac{|\mathcal{N}_{\text{SEL}}|}{|\mathcal{N}|}$
3. Continuous Execution Cost (CEC): $r_{\text{CEC}} \leftarrow r_{\text{OG}} - \lambda_2 \sum_{i \in \mathcal{N}_{\text{SEL}}} ||\mathcal{A}_{\text{SEL}_{\text{xy}}}||_2$
4. Merged Execution Cost (MEC): $r_{\text{GEC}} \leftarrow r_{\text{OG}} - \lambda_1 \frac{|\mathcal{N}_{\text{SEL}}|}{|\mathcal{N}|} - \lambda_2 \sum_{i \in \mathcal{N}_{\text{SEL}}} ||\mathcal{A}_{\text{SEL}_{\text{xy}}}||_2$

Where λ_1 and λ_2 are weighting hyperparameters that control the trade-off between task performance and resource efficiency. The policy learns that robots with $a_z = a_z^{\text{HIGH}}$ do not interact with the objects and can mitigate unnecessary collisions due to crowding. The complete loss functions and SAC algorithm details are provided in Appendix Sec. 9.2.2.

5 Experiments

We train all our policies in simulation and evaluate our proposed methods zero-shot for **DDM** tasks using the delta arrays (Sec. 3) in both MuJoCo and on the real hardware. Our evaluation follows a set of pre-defined trajectories consisting of 20 subgoals. The task is closed-loop SE(2) manipulation of 10 different objects (shown in Appendix 9.6) from an initial pose to a subgoal pose sequentially. We measure success rate (reaching a subgoal within 3 attempts) and mean trajectory tracking error ($\mu \pm \sigma$) using quasi-static closed-loop rollouts. Aggregate results are presented in Sec. 6, with object-specific details in Appendix 9.8. Specifically, we aim to answer 3 questions through our experiments:

Q1: *What is the significance of position embeddings for learning **DDM** policies?*

Transformers typically rely on position embeddings to incorporate sequential structures. Sinusoidal Position Embeddings (SPEs) use sine and cosine functions of frequencies along a sequence, which is beneficial for natural language tasks. While Rotary Position Embeddings (RoPE) mix pairs of coordinates in an outward-growing helical pattern, which is beneficial for fixed grid inputs. SCEs and RoPE are at the two ends of a spectrum from pretrained embeddings to scalable transformations, respectively. Hence, we ablate three intermediate position embeddings: (1) A locally constrained RoPE (*LCRoPE*) which zeroes out the attention activations beyond a distance threshold, (2) A learned relative embedding (*LRE*) that maps K neighbors of a robot to a fixed set of learnable angle vectors, and constructs a relative rotation matrix consistent across different robots, and (3) Use of integer position embeddings just like SCE, but without pretraining (*LE*). We follow the *MATBC-FT* pipeline to pretrain and finetune the policies with all the position embedding ablations.

Q2: *How does the selection of attention mechanisms affect the wall-clock speed and sample-efficiency of learning **DDM** policies?*

Standard transformer architectures use self-attention [8] and cross-attention [48] blocks stacked on top of each other to learn shared representations between sequential and conditional inputs. However, recent works like Dhariwal and Nichol [49], Peebles and Xie [43] have demonstrated the effectiveness of Adaptive Layer Normalization (AdaLN) to speed up the training of large autoregressive diffusion transformers. We compare AdaLN-Zero against standard self-attention and cross-attention in terms of training stability, speed of convergence using *MATSAC*.

Q3: *What are the tradeoffs of inducing policies to use fewer robots on the ability of the delta arrays to perform long-horizon closed-loop planar manipulation tasks?*

While achieving high task performance is paramount, the efficiency of execution is also critical for **DDM** systems, especially in the real world. Effectively selecting a subset of robots ($\mathcal{N}_{\text{SEL}}$) for

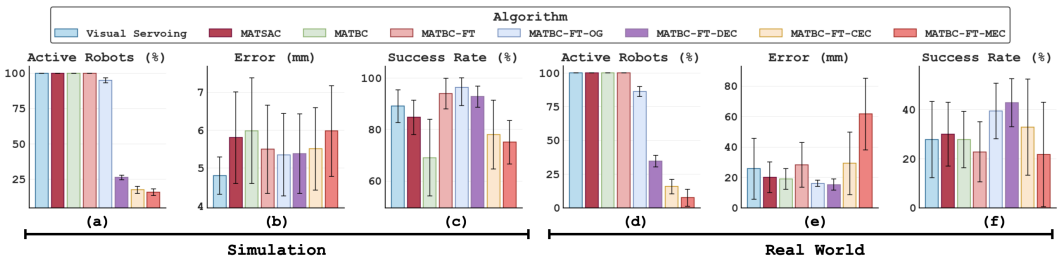


Figure 4: Overall performance for three rigid objects (hexagon, star, trapezium) on (a - c) Simulation, (d - f) Real World. For each, (Left) shows the percentage of active robots involved in each manipulation step. (Middle) shows the error at every execution step. (Right) shows the percentage success rate of reaching subgoals in 3 tries over the trajectories

a given manipulation step can reduce energy consumption, computational load during inference, physical wear, and the risk of inter-robot collisions. To study the effect of the action selection rewards in 4.4, we train four *MATBC-FT* policies: *MATBC-FT-OG*, *MATBC-FT-DEC*, *MATBC-FT-CEC*, *MATBC-FT-MEC*. We compare these methods with visual servoing, *MATBC*, *MATSAC* trained from scratch, and *MATBC-FT* with all robots selected, and quantify the tradeoff in terms of the average number of active robots selected, tracking error along the trajectory, and success rate in achieving subgoals along the trajectory. Implementation details, hardware specifications, and training protocols are in the Appendix Sec. 9.3.

6 Results and Discussion

6.1 Significance of Spatial Embeddings: Our experiments with training SPE and RoPE fail to converge while training *MATSAC* from scratch. Hence, our key ablations are performed on the *MATBC-FT-OG* pipeline. Using Fig. 8 as reference, we see that SPE still fails to learn any meaningful policies. However, RoPE shows dramatic improvement over RL training from scratch when the initial exploration is mitigated. Our ablations with *LCRoPE*, *LRE*, and *LE* resulted in poorly trained policies that fail to accomplish the trajectory object pushing task. The key takeaways are: (1) Learning position embeddings online is hard due to poor multimodal reasoning capabilities of AdaLN-Transformers. (2) Structured relative spatial bias over the entire attention mechanism is crucial for learning centralized MARL policies for **DDM** tasks. (3) Pretraining SCEs provides a highly contrastive conditioning to the attention mechanism, compared to the relatively similar weighing functions provided by RoPE.

However, they do not adequately inform the spatial arrangement of robots to condition the actions, especially when a subset of them are provided as input in arbitrary order. Unlike language and a fixed grid over images, robot selection in multi-agent settings is stochastic. We hypothesize that using predetermined embeddings to represent a dynamically varying order of the sequential input induces multi-modality in the learning pipeline, which makes it hard for policy gradient methods to converge [50, 51]. As such, policies trained using Sinusoidal Position Embeddings (SPEs) or Rotary Position Embeddings (RoPEs) failed to converge and achieve meaningful task performance (Fig. 7a).

6.2 Efficiency of Attention Mechanisms: Our results show that using AdaLN-Zero layers leads to significantly faster and more stable training compared to standard Self-Attention (SA) or Cross-Attention (CA) blocks (Fig. 7b). The wall-clock training time for AdaLN-Zero was approximately half that of SA/CA ($\sim$23 vs. $\sim$44 hours), consistent with findings in other domains [43, 20]. Furthermore, AdaLN-Zero achieved better final policy performance (higher average reward). This validates our choice of using AdaLN-Zero for computational efficiency without sacrificing performance in highly redundant **DDM** tasks.

6.3 Trajectory Tracking Performance and Action Selection Tradeoff: We evaluate the core learning pipeline and the impact of action selection strategies in both simulation and through zero-shot

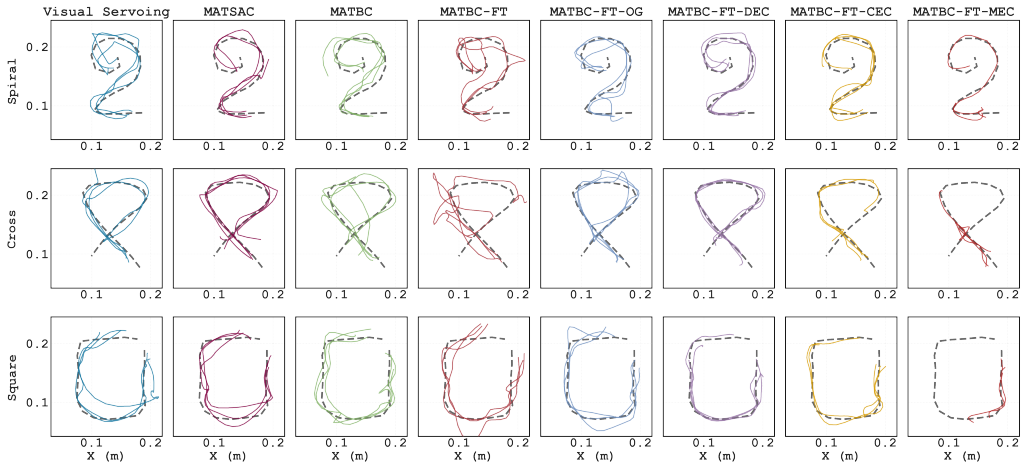


Figure 5: Trajectory tracking results on our three hardest trajectories with three objects – hexagon, trapezium, star – in the real world.

transfer to real hardware (Fig. 4). We define success as the object reaching the goal within a mean Euclidean distance of 7.5 mm over the 2D boundary points of the object. Since visual servoing actions are 2D vectors, the training data for behavior cloning are 2D vectors as well. *MATBC*, *MATSAC*, and *MATBC-FT* engage all robots in $\mathcal{N}_t \leq 64$. MATBC pretrains on the visual servoing data with a_z always to a_z^{LOW} to obtain a pretrained policy. During finetuning, our action selection mechanism learns to select $\mathcal{N}_{\text{SEL}} \subseteq \mathcal{N}_t$. Only the robots with $a_z = a_z^{\text{LOW}}$ manipulate the object, and $a_z = a_z^{\text{HIGH}}$ disengage. This reduces the active number of robots from $|\mathcal{N}_t|$ to $|\mathcal{N}_{\text{SEL}}|$. *MATBC-FT-*(*OG, DEC, CEC, MEC*) all allow learned a_z values, and thus trade off between the number of robots selected and the task performance.

RL fine-tuning (*MATBC-FT*) consistently improves upon *MATBC* alone, and *MATSAC* from scratch, which is consistent with robot learning literature [31, 52]. *MATBC-FT* achieves the lowest tracking error and highest success rate among methods using full $\mathcal{N}_t$. However, when action selection is enabled, *MATBC-FT-OG* marginally improves upon *MATBC-FT* in sim ($\sim 1.8\%$), but significantly improves in the real world ($\sim 40.7\%$) in average tracking error. We attribute this to deterministic physics in sim causing the performance improvement to saturate. Moreover, comparing *MATBC-FT-OG* with *MATBC-FT-DEC*, *MATBC-FT-CEC*, and *MATBC-FT-MEC*, we conclude that adding a discrete penalty to the reward formulation encourages MATs to better leverage the tradeoff between execution cost and planar manipulation performance. Fig. 5 shows that *MATBC-FT-CEC* and *MATBC-FT-MEC* really struggle with closed-loop long-horizon **DDM** tasks. On the other hand, *MATBC-FT-DEC* faithfully tracks objects along the desired trajectory, while using $\sim 55\%$ fewer robots compared to *MATBC-FT-OG*, and $\sim 65\%$ fewer than *MATBC-FT*.

6.4 Qualitative Evaluation: We observe significantly high policy performance in deterministic settings, as are simulation environments, compared to the stochasticity of soft robot kinematics in the real world. The soft delta robots in the delta arrays demonstrate a spring-damper behaviour when multiple robots manipulate a single object. This makes execution in MuJoCo highly stable, but creates practical challenges in the real world due to the making and breaking of compliant contacts induce out-of-distribution errors during trajectory tracking for the objects. Hence we see a significant drop in success rates of *MATBC-FT* and *MATBC-FT-OG*, which use a higher number of robots, compared to the relatively minor drop of *MATBC-FT-DEC*. This can also be qualitatively seen in Fig. 12 and Fig. 5. We further show out-of-distribution object generalization by deploying the full suite of methods on a push-T task in Fig. 10

7 Conclusion

In this work, we presented an initial exploration toward learning robust and sample-efficient policies for Distributed Dexterous Manipulation (**DDM**) on delta arrays using spatially conditioned MAT. We demonstrate how design choices like attention mechanisms and position embeddings significantly affect the performance of training multi-agent SAC using MATs. Further, we demonstrate that incorporating an action selection mechanism with a discrete penalty (*MATBC-FT-DEC*) enables the policy to learn an effective policy that maximizes performance and efficiency compared to continuous penalties. *MATBC-FT-DEC* reduces active robot usage by up to 65% while maintaining ~ 1.5 cm average trajectory tracking error with a soft robot array. Moreover, action selection significantly mitigates robotic wear and tear in real-world deployment.

8 Limitations

In this paper, we proposed multi-agent policy learning for DDM as a novel problem formulation. Hence, we limited the difficulty of tasks to 2D planar manipulation. Studying the efficiency of these methods to learn DDM policies in 3D space remains an unexplored problem. On the other hand, although fine-tuning shows a major improvement in sample efficiency, it still needs expert demonstrations to pretrain the behavior cloning policy, which is challenging for the multi-robot cooperation domain. Expert demonstrations will be harder to obtain in 3D space. We identify tabula rasa training not being able to match the performance of fine-tuned methods as the second limitation. Moreover, we can show our experiments only on the delta arrays, as the soft robots allow for such experimental setups to be made possible. This can be considered a limitation in terms of the demonstration of the generalizability of the proposed method for **DDM** tasks. Diffusion models have been extensively used in behavior cloning, but RL-based fine-tuning of multi-agent diffusion policies remains an unexplored avenue, especially for highly redundant action spaces as those of the delta arrays. Finally, in this paper, we explored the use of multi-agent transformers for **DDM** tasks, which can be thought of as a centralized method for policy learning. Understanding the fundamental differences in decentralized vs centralized methods for distributed dexterous manipulation can a fundamental area of future work.

Acknowledgments

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

[1] J. E. Luntz, W. Messner, and H. Choset. Distributed manipulation using discrete actuator arrays. *The International Journal of Robotics Research*, 20(7):553–583, 2001.

[2] K. F. Böhringer and H. Choset. *Distributed manipulation*. Springer Science & Business Media, 2000.

[3] M. Nakashige, K. Hirota, and M. Hirose. Linear actuator for high-resolution tactile display. In *RO-MAN 2004. 13th IEEE International Workshop on Robot and Human Interactive Communication (IEEE Catalog No. 04TH8759)*, pages 587–590. IEEE, 2004.

[4] S. Follmer, D. Leithinger, A. Olwal, A. Hogge, and H. Ishii. inform: dynamic physical affordances and constraints through shape and object actuation. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, page 417–426, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450322683. doi: 10.1145/2501988.2502032. URL https://doi.org/10.1145/2501988.2502032.

[5] S. Patil, T. Tao, T. Hellebrekers, O. Kroemer, and F. Z. Temel. Linear delta arrays for compliant dexterous distributed manipulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10324–10330, 2023. doi:10.1109/ICRA48891.2023.10160578.

[6] T. D. Murphey, J. W. Burdick, J. Burgess, and A. Homyk. Experiments in nonsmooth control of distributed manipulation. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 3, pages 3600–3606. IEEE, 2003.

[7] Z. Xue, H. Zhang, J. Cheng, Z. He, Y. Ju, C. Lin, G. Zhang, and H. Xu. Arraybot: Reinforcement learning for generalizable distributed manipulation through touch. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16744–16751. IEEE, 2024.

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

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[10] M. Yim, J. Reich, and A. A. Berlin. Two approaches to distributed manipulation. In *Distributed Manipulation*, pages 237–261. Springer, 2000.

[11] M. B. Cohn, K. F. Boehringer, J. M. Noworolski, A. Singh, C. G. Keller, K. A. Goldberg, and R. T. Howe. Microassembly technologies for mems. In *Microelectronic Structures and MEMS for Optical Processing IV*, volume 3513, pages 2–16. SPIE, 1998.

[12] G. Pangaro, D. Maynes-Aminzade, and H. Ishii. The actuated workbench: computer-controlled actuation in tabletop tangible interfaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 181–190, 2002.

[13] D. Leithinger, D. Lakatos, A. DeVincenzi, M. Blackshaw, and H. Ishii. Direct and gestural interaction with relief: a 2.5 d shape display. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 541–548, 2011.

[14] B. R. Donald, J. Jennings, and D. Rus. Information invariants for distributed manipulation. *The International Journal of Robotics Research*, 16(5):673–702, 1997. doi:10.1177/027836499701600506. URL https://doi.org/10.1177/027836499701600506.

[15] K. Böhringer, R. Brown, B. Donald, J. Jennings, and D. Rus. Distributed robotic manipulation: Experiments in minimalism. In O. Khatib and J. K. Salisbury, editors, *Experimental Robotics IV*, pages 11–25, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[16] B. Donald, L. Gariepy, and D. Rus. Experiments in constrained prehensile manipulation: Distributed manipulation with ropes. In *Experimental Robotics VI*, pages 25–36, London, 2000. Springer London.

[17] Q. Wang, R. McCarthy, D. C. Bulens, F. R. Sanchez, K. McGuinness, N. E. O'Connor, and S. J. Redmond. Identifying expert behavior in offline training datasets improves behavioral cloning of robotic manipulation policies. *IEEE Robotics and Automation Letters*, 2023.

[18] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[19] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[20] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[21] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[22] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[23] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[24] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[27] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 29–31 Oct 2018.

[28] S. Noh, S. Kim, and I. Jang. Efficient fine-tuning of behavior cloned policies with reinforcement learning from limited demonstrations. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

[29] S. Dasari, J. Wang, J. Hong, S. Bahl, Y. Lin, A. Wang, A. Thankaraj, K. Chahal, B. Calli, S. Gupta, et al. Rb2: Robotic manipulation benchmarking with a twist. *arXiv preprint arXiv:2203.08098*, 2022.

[30] B. Jia and D. Manocha. Sim-to-real robotic sketching using behavior cloning and reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 18272–18278. IEEE, 2024.

[31] R. Ramrakhya, D. Batra, E. Wijmans, and A. Das. Pirlnav: Pretraining with imitation and rl finetuning for objectnav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17896–17906, 2023.

[32] X. Wang, H. Xu, Y. Zheng, and X. Zhan. Offline multi-agent reinforcement learning with implicit global-to-local value regularization. *Advances in Neural Information Processing Systems*, 36:52413–52429, 2023.

[33] O. Nachum, M. Ahn, H. Ponte, S. Gu, and V. Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

[34] F. Jakob, F. Gregory, A. Triantafyllos, N. Nantas, and W. Shimon. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[35] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4295–4304, 2018.

[36] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5887–5896, 2019.

[37] Y. Chen, Y. Geng, F. Zhong, J. Ji, J. Jiang, Z. Lu, H. Dong, and Y. Yang. Bi-dexhands: Towards human-level bimanual dexterous manipulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[38] M. Wen, J. G. Kuba, R. Lin, W. Zhang, Y. Wen, J. Wang, and Y. Yang. Multi-agent reinforcement learning is a sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=1W8UwXAQubL.

[39] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[40] Y. Chebotar, Q. Vuong, K. Hausman, F. Xia, Y. Lu, A. Irpan, A. Kumar, T. Yu, A. Herzog, K. Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pages 3909–3928. PMLR, 2023.

[41] P. Mannam, A. Rudich, K. L. Zhang, M. Veloso, O. Kroemer, and Z. Temel. A low-cost compliant gripper using cooperative mini-delta robots for dexterous manipulation. In *Robotics science and systems*, 2021.

[42] P. Vischer, R. Clavel, et al. Kinematic calibration of the parallel delta robot. *Robotica*, 16(2): 207–218, 1998.

[43] W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[44] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[45] P. Goyal. Accurate, large minibatch sg d: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[46] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1290–1299, 2022.

[47] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[48] C.-F. R. Chen, Q. Fan, and R. Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 357–366, 2021.

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

[50] Z. Huang, L. Liang, Z. Ling, X. Li, C. Gan, and H. Su. Reparameterized policy learning for multimodal trajectory optimization. In *International Conference on Machine Learning*, pages 13957–13975. PMLR, 2023.

[51] S. Li, R. Krohn, T. Chen, A. Ajay, P. Agrawal, and G. Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient. *Advances in Neural Information Processing Systems*, 37:38456–38479, 2024.

[52] K. Xu, Z. Hu, R. Doshi, A. Rovinsky, V. Kumar, A. Gupta, and S. Levine. Dexterous Manipulation from Images: Autonomous Real-World RL via Substep Guidance. Technical report, Dec. 2022. URL http://arxiv.org/abs/2212.09902. arXiv:2212.09902 [cs] type: article.

[53] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[54] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

9 Appendix

9.1 Visual Servoing

Providing human demonstrations for a 64-robot system is non-trivial due to the complexity of coordinating many agents. Hence, we propose a visual servoing pipeline to collect expert demonstrations $\mathcal{D}_{exp}$ for *MATBC* pre-training (Sec. 4.3).

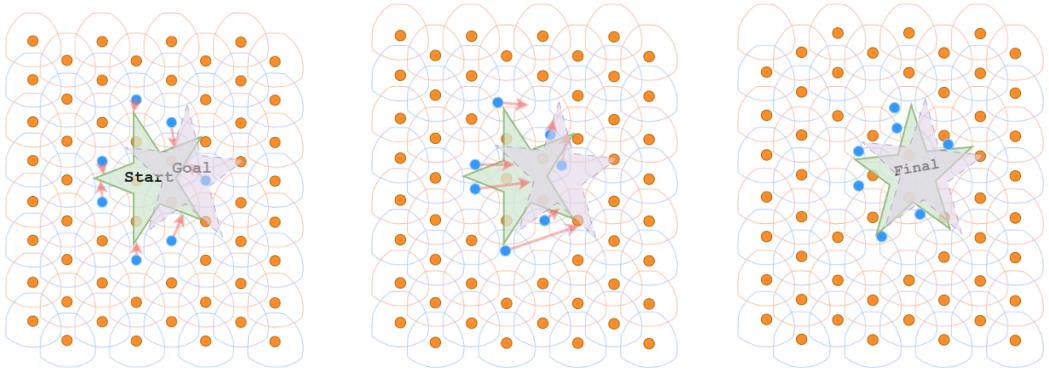
First, we segment objects in the camera view using Language Segment Anything Model (Lang-SAM) [53, 54] for complex shapes or standard HSV filtering for simpler geometries. This yields a set of M 2D boundary points $\mathcal{B} = \{b_1, b_2, \ldots, b_M\} \in \mathbb{R}^{M\times 2}$. We then run a nearest neighbor search algorithm to identify the set of robots $\mathcal{N}$ whose workspaces overlap with the object's boundary but whose base positions are outside the object's initial polygon. For each robot $i \in \mathcal{N}$, we identify its closest boundary point $b_i \in \mathcal{B}$.

For a given target pose (defined by a relative 2D translation and rotation), we compute the corresponding 2D rigid transformation matrix T. We compute the goal boundary points $\mathcal{B}'$ by applying this transformation to the initial boundary points: $\mathcal{B}' = T(\mathcal{B}) = \{T(b_1), T(b_2), \ldots, T(b_M)\}$. The corresponding goal point for robot i's closest initial point b_i is $b'_i = T(b_i)$.

The raw visual servoing action a^e_i for robot i is generated as the displacement vector (flow vector) from its corresponding initial boundary point b_i to the goal boundary point b'_i:

$$a^e_i = \text{clip}(b'_i - b_i, a_{\text{min}}, a_{\text{max}}) \tag{2}$$

where $a_{\text{min}}, a_{\text{max}}$ are the workspace limits of the delta robots.



(a) Grasping (b) Pushing (c) Final

Figure 6:

9.2 Additional Loss Function Details

9.2.1 Behavior Cloning (BC) Loss Functions

The *MATBC* policy, π^{BC}_ϕ, is trained by minimizing the Mean Squared Error (MSE) between the predicted actions $a^\pi_{t,i} = \pi^{BC}_\phi(s_t)_i$ and the expert actions $a^e_{t,i}$ for each agent i in the neighborhood $\mathcal{N}_t$:

$$J_{BC}(\phi) = \mathop{\mathbb{E}}_{(s_t, a^e_t)\sim\mathcal{D}_{exp}} \left[\sum_{i\in\mathcal{N}_t} \frac{1}{2} ||a^\pi_{t,i} - a^e_{t,i}||^2_2 \right] \tag{3}$$

Note: $a^e_t = [a^e_{t,1}, \ldots, a^e_{t,N_t}]$ is the joint expert action. $a^\pi_{t,i}$ is the action predicted for agent i by the policy given the joint state s_t.

The initial critic network Q^{BC}_θ used for BC pre-training (Sec. 4.3) is trained to predict the immediate global reward $r_t = r(s_t, a_t)$ using mean squared error loss on the mixed dataset $\mathcal{D}_{mixed}$. Assuming

a centralized critic predicting the global reward:

$$J_{Q^{BC}}(\theta) = \mathop{\mathbb{E}}_{(s_t,a_t,r_t)\sim\mathcal{D}_{mixed}} \left[\frac{1}{2} \left(Q_\theta^{BC}(s_t, a_t) - r_t \right)^2 \right] \tag{4}$$

Note: $Q_\theta^{BC}(s_t, a_t)$ represents the predicted immediate global reward given the joint state s_t and joint action a_t. (If the critic is intended to output per-agent values summing to r_t, the equation would need adjustment).

9.2.2 Multi-Agent Soft Actor Critic Loss Functions

Assuming the critic decomposes the global value into per-agent Q-values $Q_{\theta_k}(s_t, a_t)_i$ for $i \in \mathcal{N}_t$, the critic loss for each of the two critics ($k = 1, 2$) is defined as:

$$J_Q(\theta_k) = \mathop{\mathbb{E}}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}} \left[\sum_{i\in\mathcal{N}_t} \frac{1}{2} \left(Q_{\theta_k}(s_t, a_t)_i - \hat{q}_{t,i} \right)^2 \right] \quad \text{for } k = 1, 2 \tag{5}$$

where $\mathcal{D}$ is the replay buffer containing environment interaction data, and the target Q-value $\hat{q}_{t,i}$ for agent i at time t is:

$$\hat{q}_{t,i} = r_t + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi_\phi(\cdot|s_{t+1})} \left[\min_{k=1,2} Q_{\bar{\theta}_k}(s_{t+1}, a_{t+1})_i - \alpha_{ent} \log \pi_\phi(a_{t+1,i}|s_{t+1}) \right] \tag{6}$$

Here, $\bar{\theta}_k$ are the target network parameters (updated via Exponential Moving Average (EMA)), γ is the discount factor, and α_{ent} is the entropy temperature. $a_{t+1,i}$ is the action for agent i in the next joint action a_{t+1}.

The actor loss is defined as:

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{s_t\sim\mathcal{D},\tilde{a}_t\sim\pi_\phi(\cdot|s_t)} \left[\sum_{i\in\mathcal{N}_t} \left(\alpha_{ent} \log \pi_\phi(\tilde{a}_{t,i}|s_t) - \min_{k=1,2} Q_{\theta_k}(s_t, \tilde{a}_t)_i \right) \right] \tag{7}$$

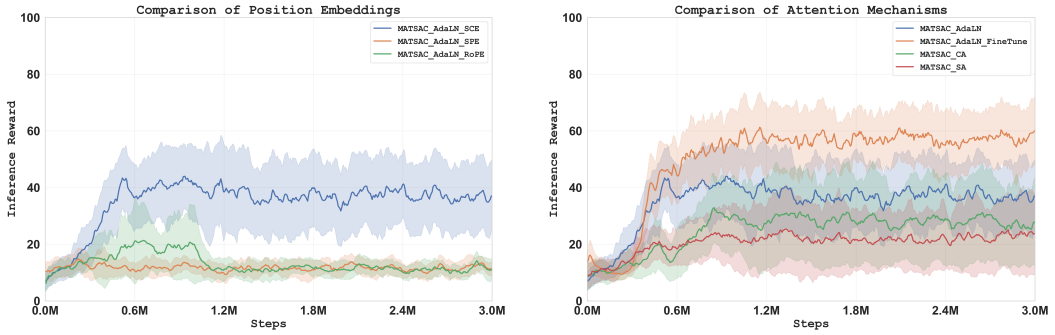
where $\tilde{a}_t = [\tilde{a}_{t,1}, \ldots, \tilde{a}_{t,N_t}]$ are actions sampled from the Squashed Gaussian policy [9] using the reparameterization trick for differentiability. $\pi_\phi(\tilde{a}_{t,i}|s_t)$ represents the probability of agent i's action under the policy given the joint state s_t.

9.3 Training Protocol and Hardware Specifications

We use 10-layer MATs for the policy and two Q-values, respectively, with 128-dim weights for all hidden layers. For MABC, visual servoing, as *expert demonstrations*, is collected for 500 episodes per object in sim, and the pretraining phase runs for 400 epochs to obtain *MATBC*. All *MATSAC* policies (including fine-tuning and training from scratch) are trained for 3,000,000 environment steps with a batch size of 256 and standard learning rates of 3e-4, discount factor $\gamma = 0.99$. We use an Nvidia 4090 with an AMD EPYC 9554 CPU for all our experiments.

9.4 Ablation Results

The following plots show the learning curves while training *MATSAC* from scratch.



(a) (b)

Figure 7: **Inference Reward Plots** a) Attention Mechanisms - AdaLN performs better than Cross Attention and Self Attention. AdaLN transformer fine-tuned using SAC performs better than training tabula-rasa. b) Position Embeddings - Sinusoidal and Rotary Embeddings fail to learn spatial correlations among neighboring sets of robots while SCEs demonstrably do so.

The following plots show the performance of *MATBC-FT-OG* with all the position embedding ablations.

9.5 Position Embedding Ablations

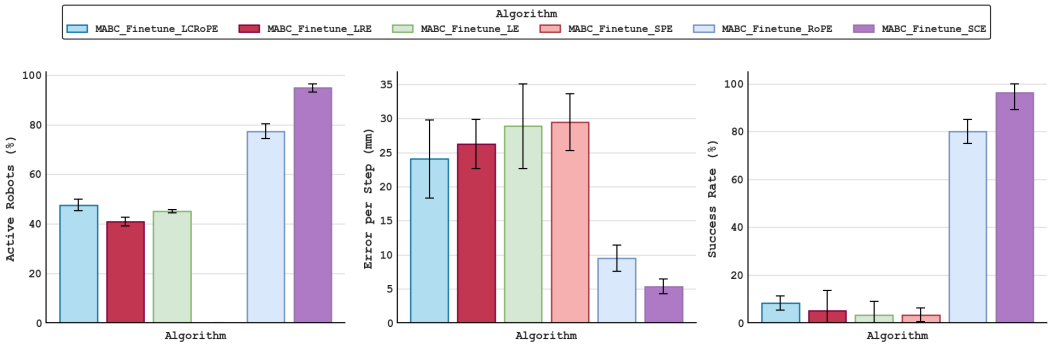


Figure 8:

9.6 Experiment Trajectories

Ground truth trajectories which we manipulate the objects over:

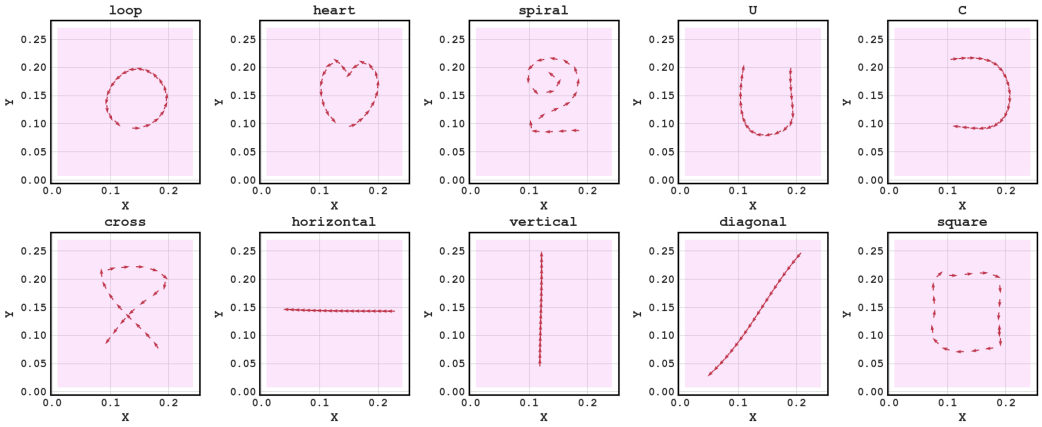


Figure 9: Ground truth trajectories defined as the inference task for closed-loop **DDM** tasks.

9.7 Object Level Performance - Real

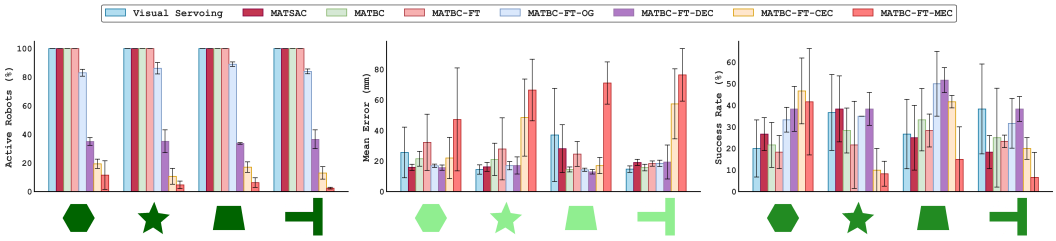
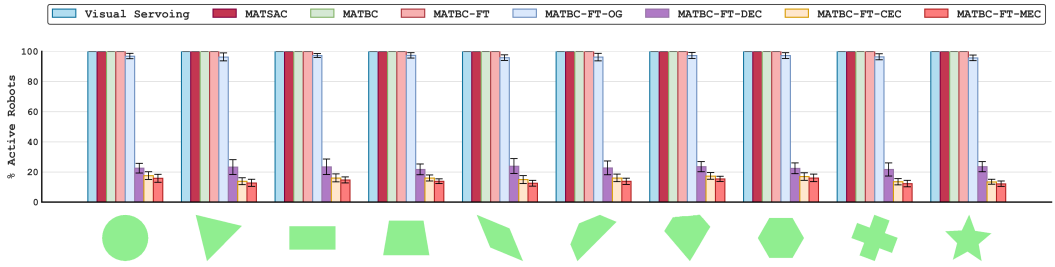


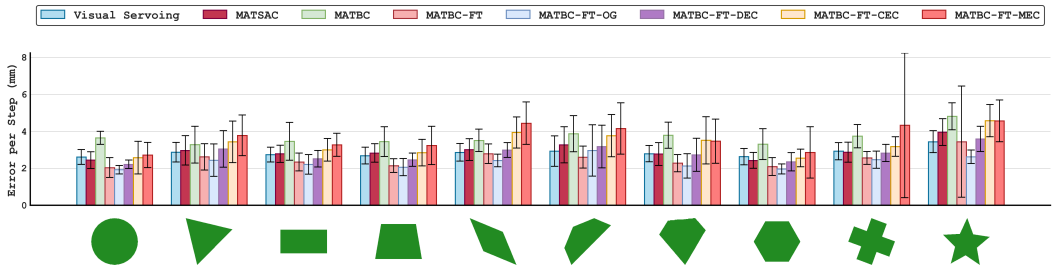
Figure 10: Object level comparison between all the methods in the real world. The T-shaped object was unseen during training. (a) Shows percentage of active robots involved in each manipulation step. (b) Shows the error at every execution step. (c) Shows the number of attempts needed by an algorithm to complete a 20-step trajectory.

9.8 Object Level Performance - Sim

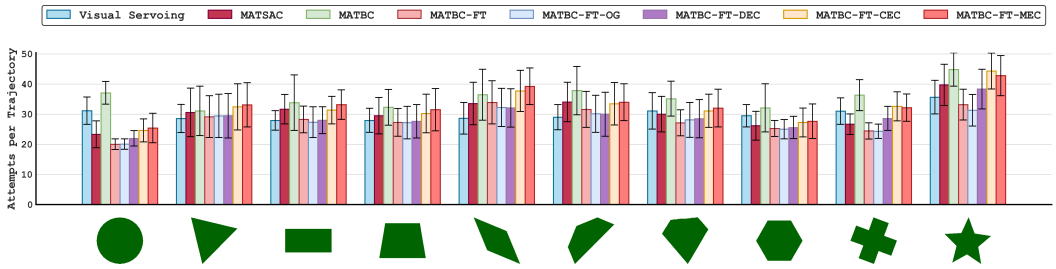
We show the performance of all algorithms on all objects averaged over all trajectories in simulation. The trends show that generally, convex objects are easier to manipulate by the delta arrays due to their workspace limitations, but they struggle with non-convex objects, especially during larger rotation angles.



(a)



(b)



(c)

Figure 11: Object level comparison between all the methods. (a) shows percentage of active robots involved in each manipulation step. (b) shows the error at every execution step. (c) shows the number of attempts needed by an algorithm to complete a 20-step trajectory.

9.9 Results Per Trajectory

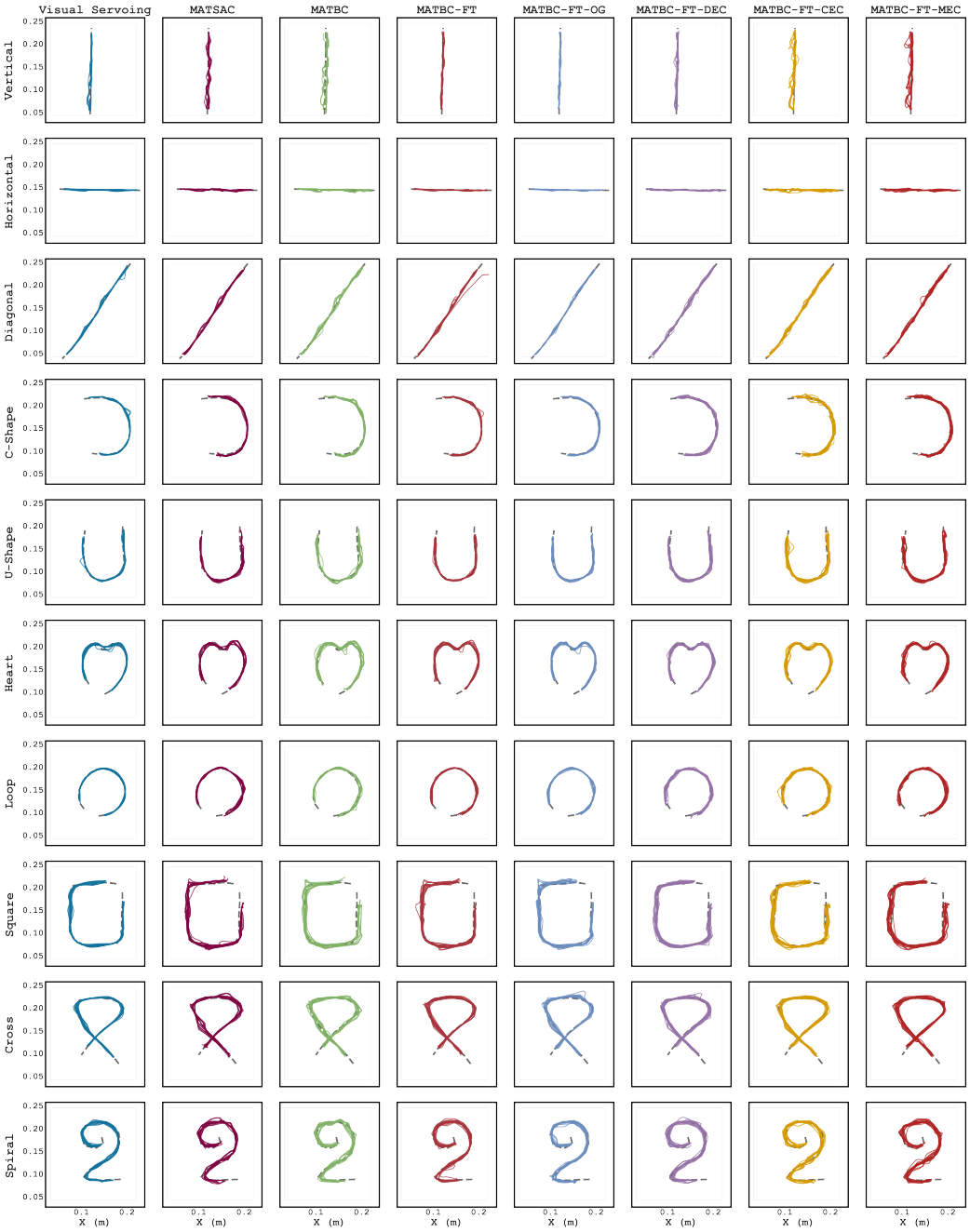


Figure 12: Comparison between different methods while tracking objects being manipulated along all target trajectories. For each subplot, we collect data over all 10 objects and run 5 trials in simulation.

10 Spatial Contrastive Embeddings Ablations:

10.1 Locally Constrained RoPE (LC-RoPE)

We ablate the standard RoPE formulation to incorporate spatial locality constraints based on the physical arrangement of delta robots. We introduce a locality mask M_{ij} that constrains attention to spatially proximate robots within the workspace overlap radius $r_{\text{local}} = 6$ cm after the Qeury and Key matrix multiplication in the RoPE attention mechanism:

$$M_{ij} = \begin{cases} 1 & \text{if } d_{ij} \leq r_{\text{local}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The attention weights are computed with spatial locality constraints to ensure attention is restricted to spatially local neighborhoods:

$$A_{ij} = M_{ij} \cdot \frac{\exp\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d_k}}\right)}{\sum_{l \in \mathcal{N}_t} M_{il} \cdot \exp\left(\frac{\mathbf{q}_i^T \mathbf{k}_l}{\sqrt{d_k}}\right)} \tag{9}$$

10.2 Learned Relative Embeddings (LRE)

This ablation replaces the fixed-angle calculation in RoPE with a learned lookup Embedding table corresponding to the relative neighborhoods with respect to the geometry of the hexagonal arrangement of the delta arrays. We categorize embeddings into a finite set of neighborhood classes. For our work, we consider $K = 13$ such classes: the robot's own position, its six immediate neighbors, and its six intermediate-nearest neighbors.

10.2.1 Learnable Angle Lookup Table

We define a learnable lookup table (nn.embedding), $\mathbf{\Phi}$, which stores the rotation angles for each of these K relative position classes.

$$\mathbf{\Phi} \in \mathbb{R}^{K \times d/2} \tag{10}$$

Here, d is the model's embedding dimension. This matrix maps each of the K spatial classes to a vector of $d/2$ angles, one for each 2D subspace, mirroring the structure of RoPE. This lookup table is the core learnable component of LRE.

10.2.2 Mapping and Angle Retrieval

For any two robots, i and j, located at grid positions (g_x^i, g_y^i) and (g_x^j, g_y^j), we first determine their relative position class, c_{ij}:

$$c_{ij} = \rho\left((g_x^j - g_x^i, g_y^j - g_y^i)\right) \in \{0, 1, \ldots, K-1\} \tag{11}$$

The function ρ is a deterministic mapping that takes a 2D displacement vector on the hexagonal grid and returns its corresponding integer class index. We then use this index to retrieve the learned angle vector $\boldsymbol{\phi}_{ij}$ from the lookup table:

$$\boldsymbol{\phi}_{ij} = \mathbf{\Phi}[c_{ij}] = [\phi_{ij,1}, \phi_{ij,2}, \ldots, \phi_{ij,d/2}] \tag{12}$$

10.2.3 Relative Rotation Matrix Construction

Using the retrieved angles, we construct a block-diagonal rotation matrix $\mathbf{R}_{ij}$ that represents the relative spatial transformation from robot i to robot j.

$$\mathbf{R}_{ij} = \text{BlockDiag}\left(\mathbf{R}_{ij}^{(1)}, \mathbf{R}_{ij}^{(2)}, \ldots, \mathbf{R}_{ij}^{(d/2)}\right) \tag{13}$$

Each 2x2 rotation block $\mathbf{R}_{ij}^{(k)}$ is a standard rotation matrix formulated using the corresponding learned angle $\phi_{ij,k}$:

$$\mathbf{R}_{ij}^{(k)} = \begin{pmatrix} \cos(\phi_{ij,k}) & -\sin(\phi_{ij,k}) \\ \sin(\phi_{ij,k}) & \cos(\phi_{ij,k}) \end{pmatrix} \tag{14}$$

This construction is directly analogous to the rotation matrix used in the original RoPE formulation.

10.2.4 Application in Self-Attention

Finally, we incorporate this learned relative rotation into the self-attention mechanism. After projecting the input embeddings $\mathbf{x}_i$ and $\mathbf{x}_j$ into queries and keys, we apply the rotation.

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \tag{15}$$

$$\mathbf{k}_j = \mathbf{W}_k \mathbf{x}_j \tag{16}$$

The attention score between robots i and j is computed by applying the relative rotation matrix $\mathbf{R}_{ij}$ to the key vector before the dot product with the query vector:

$$\text{score}(i, j) = \frac{(\mathbf{q}_i)^T (\mathbf{R}_{ij} \mathbf{k}_j)}{\sqrt{d}} \tag{17}$$

This formulation ensures that the attention score is modulated by a learned geometric relationship, directly injecting the spatial structure of the hexagonal grid into the self-attention mechanism. By making the rotation angles in $\mathbf{\Phi}$ learnable, our model can determine the optimal transformations for each spatial relationship through gradient descent, rather than relying on a fixed mathematical formula that may not perfectly capture the underlying physics of the multi-robot system.