MILD-MPC: MULTI-AGENT IMITATION LEARNING VIA DISTRIBUTED MODEL PREDICTIVE CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Controlling multi-agent systems in real time requires a scalable optimization method that satisfies constraints. However, existing distributed trajectory optimization methods typically assume fixed problem structures, and their solve times exceed the time budget as the number of agents grows. We propose **Multi-agent Imitation Learning via Distributed Model Predictive Control (MILD-MPC)**, a framework for learning a permutation invariant predictive control policy that imitates a distributed MPC expert, generalizes to larger problems, and runs in a single forward pass at deployment. In addition, we develop Model Predictive Distributed Differential Dynamic Programming (MPD-DDP), a new distributed MPC method that handles time-varying problem structures, provides high-quality expert demonstrations, and serves as a backup policy when a safety verifier invalidates the learned policy's trajectory. We demonstrate that the deployed policy achieves substantial timing improvements over both a centralized MPC baseline and the MPD-DDP expert, generalizes to problems with more agents, and remains robust to agent failures during deployment, across 2D and 3D obstacle-field navigation tasks. In hardware experiments, a policy trained with 8 agents transfers to 15 robots without retraining, completing the task collision-free in real time.

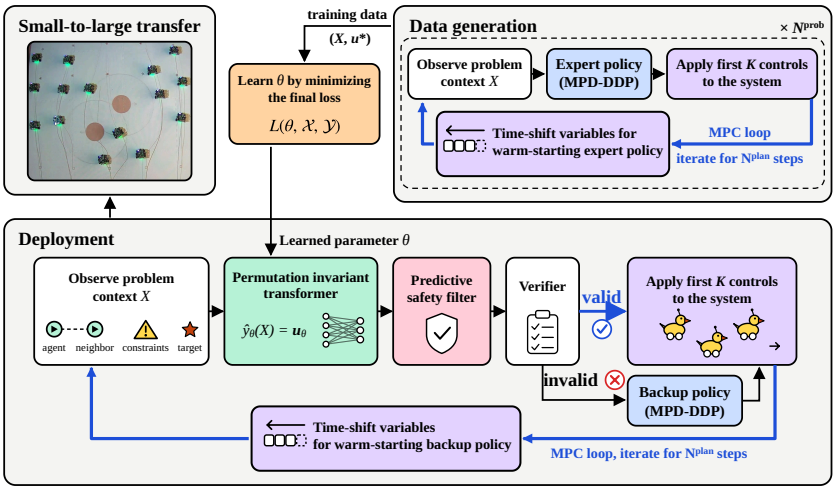


Figure 1: Multi-agent Imitation Learning via Distributed Model Predictive Control (MILD-MPC) framework overview. *Data generation*: The dataset is generated using the expert policy (MPD-DDP) in a receding horizon fashion. *Learning*: The loss function is formulated to minimize the constraint residuals and the trajectory discrepancies between the expert and the learned policy under known-transition setting. *Deployment*: Given the learned parameter θ, each agent predicts the control trajectory in receding horizon using the learned policy parameterized by permutation invariant Transformer. The learned policy's control trajectory is refined through the multi-agent predictive safety filter. If the predicted trajectory is validated by the verifier, then its first K controls are directly applied to the system. Otherwise, all agents are routed to use the backup policy (MPD-DDP). *Small-to-large transfer*: The learned policy generalizes to larger teams at deployment.

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

1 INTRODUCTION

Autonomous systems are increasingly deployed in the real world, and many operate as teams. However, scaling multi-agent control is challenging because high-fidelity data is expensive to collect, physical safety constraints need to be satisfied, and real-time operation favors a lightweight model. Under these conditions, the following requirements are difficult to meet at once: (i) scaling to many agents, (ii) operating within a real-time budget, and (iii) ensuring state and control constraints.

To improve the scalability of multi-agent control under constraints, several consensus ADMM-based distributed trajectory optimization methods have been proposed (Halsted et al., 2021). Saravanos et al. (2023) develop scalable distributed optimization architectures extending differential dynamic programming (DDP), but assume a time-invariant neighborhood set, and the real time requirement is not addressed. Huang et al. (2023) propose a decentralized iLQR method that targets real-time performance, yet assume a complete communication graph.

To control multi-agent systems in real time, distributed model predictive control (MPC) serves as an effective framework (Camponogara et al., 2002; Maestre et al., 2014). Luis et al. (2020) achieve online real-time replanning for quadrotor swarms, but assume linear dynamics and the success rates degrade as the number of agents increases. Saravanos et al. (2024) introduce distributed covariance steering combined with MPC, which relies on probabilistic constraints. Liu et al. (2024) propose an improved consensus ADMM for cooperative motion planning with a graph evolution strategy under receding horizon control, yet the subgraph structure is updated only between episodes rather than within the horizon. Zhou et al. (2026) develop a real-time distributed MPC framework for collaborative loco-manipulation by exploiting the star-shaped coupling, which does not generalize inter-agent constraints. Common to these approaches is that all computation happens online. Warm-starting reuses the preceding solves, yet the recurring structure across similar tasks is not captured.

Imitation learning (IL) trains a policy to fit expert demonstrations (Osa et al., 2018), shifting the expert's cost from deployment to training. Applied to MPC, an optimizer becomes the expert, and the learned policy costs only a forward pass at deployment (Levine & Koltun, 2013; Hertneck et al., 2018). In multi-agent settings, Riviere et al. (2020); Tsukamoto & Chung (2021) imitate a centralized optimizer to obtain a decentralized policy that runs online. However, generating demonstrations with a centralized optimizer becomes expensive as the number of agents grows, since the centralized problem scales poorly in the joint state space. Moreover, the learned policies are reactive, and thus safety depends on a barrier function or a tracking certificate rather than re-solving the problem. In a similar vein, Tolstaya et al. (2020); Gama et al. (2022) learn decentralized controllers using GNN and imitation learning, but still rely on a centralized expert controller. Unlike the aforementioned methods, Zhang et al. (2025b) jointly learn a graph control barrier function and a distributed policy, which generalizes from a small number of agents to larger problems, but there is no cooperation between controlled agents.

Distributed MPC and learning are complementary: MPC handles constraints by construction, but its solve time limits real-time scalability, while a learned policy runs in constant time at deployment without inheriting constraint satisfaction. Zhang et al. (2025a) exploit this by learning distributed MPC policies online via approximate dynamic programming, scaling to thousands of robots in simulation. However, the learned actor policy consists of a set of per-robot weight matrices tied to a predefined neighborhood, refined online rather than trained offline, and they assume a time-invariant communication network, which the authors identify as a limitation when robots join or leave.

We propose a multi-agent imitation learning framework in which a distributed MPC solver serves both roles: a scalable expert during training and a verified backup policy at deployment. For this purpose, we develop a new distributed nonlinear MPC optimizer that handles time-varying neighborhoods and enables scalable data generation unlike prior multi-agent imitation learning, which relies on centralized experts whose costs grow with the number of agents. Furthermore, the learned policy, parameterized by a permutation invariant Transformer, generalizes from a small team in training to larger teams at deployment. Our contributions are summarized as follows.

1. **Multi-agent Imitation Learning via Distributed Model Predictive Control (MILD-MPC) framework:** We propose a new framework in which the predictive learned policy imitates a distributed MPC expert, and the expert serves as a backup policy at deployment.

2. **Distributed MPC method:** We introduce Model Predictive Distributed DDP (MPD-DDP), a scalable distributed nonlinear MPC method that supports time-varying neighborhoods.
3. **Relaxed safety module:** We formulate the multi-agent predictive safety filter problem to provide a relaxed safety layer for the learned policy.
4. **Robustness, generalization, and acceleration:** In simulations, we show that the proposed method achieves a significant speedup over the baseline, generalizes in small-to-large problem transfer, and remains robust in agent failure scenarios in obstacle-field navigation tasks.
5. **Hardware test:** We demonstrate the real world applicability of the proposed method by deploying a policy trained with 8 agents to 15 robots in real time without retraining.

2 Multi-Agent Optimal Control Problem

Consider the multi-agent system with M agents, governed by the discrete-time nonlinear dynamics. The mathematical formulation of the multi-agent optimal control problem is represented as follows:

Problem 1 (Multi-agent optimal control problem)

$$\min_{\boldsymbol{u}} \sum_{i=1}^{M} J_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \sum_{i=1}^{M} \left[\Phi_i(\boldsymbol{x}_{i,T}) + \sum_{k=0}^{T-1} l_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}) \right] \tag{1}$$

$$\textit{subject to:} \quad \boldsymbol{x}_{i,k+1} = \boldsymbol{F}_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}), \qquad \forall i \in [M],\ \forall k = 0, \dots, T-1 \tag{2}$$

$$\boldsymbol{b}_i(\boldsymbol{u}_{i,k}) \leq 0, \qquad \forall i \in [M],\ \forall k = 0, \dots, T-1 \tag{3}$$

$$\boldsymbol{g}_i(\boldsymbol{x}_{i,k}) \leq 0, \qquad \forall i \in [M],\ \forall k = 0, \dots, T \tag{4}$$

$$\boldsymbol{h}_{i,j}(\boldsymbol{x}_{i,k}, \boldsymbol{x}_{j,k}) \leq 0, \qquad \forall i \in [M],\ \forall j : i < j \leq M,\ \forall k = 0, \dots, T \tag{5}$$

where $l_i : \mathbb{R}^{n_i} \times \mathbb{R}^{m_i} \to \mathbb{R}$ is the local stage cost function, $\Phi_i : \mathbb{R}^{n_i} \to \mathbb{R}$ is the local terminal cost function, each agent i has the state constraint function $\boldsymbol{g}_i : \mathbb{R}^{n_i} \to \mathbb{R}$, control constraint function $\boldsymbol{b}_i : \mathbb{R}^{m_i} \to \mathbb{R}$, and each pair of agents $i \neq j$ has the inter-agent state constraint function $\boldsymbol{h}_{i,j} : \mathbb{R}^{n_i} \times \mathbb{R}^{n_j} \to \mathbb{R}$. In robotics, for example, $\boldsymbol{b}_i$, $\boldsymbol{g}_i$, and $\boldsymbol{h}_{i,j}$ may be used to represent the constraints for the actuation limits, obstacle avoidance, and inter-agent collision avoidance.

2.1 Distributed Optimal Control Architecture

It is known that solving Problem 1 in a centralized manner scales poorly as the computational complexity increases cubically with the dimensionality of the decision variables (Nocedal & Wright, 2006; Yakowitz & Rutherford, 1984). In this section, we introduce a distributed optimal control problem that allows more scalable computation.

In multi-agent systems, agents are often coupled through constraints, which we model using a neighborhood set denoted as $\mathcal{N}_i \subseteq [M]$ for agent i. We define a neighbor-of set as $\mathcal{P}_i = \{j \in [M] : i \in \mathcal{N}_j\}$, *i.e.*, the set of all agents that have agent i as their neighbor. Each agent i has the local state $\boldsymbol{x}_{i,k} \in \mathbb{R}^{n_i}$ and local control $\boldsymbol{u}_{i,k} \in \mathbb{R}^{m_i}$ at the time instant k, where n_i and m_i are the state and control dimensions of agent i. The state and control trajectory of agent i over a planning horizon $k = 0, \dots, T$ are denoted as $\boldsymbol{x}_i = [\boldsymbol{x}_{i,0}, \dots, \boldsymbol{x}_{i,T}] \in \mathbb{R}^{n_i \times (T+1)}$ and $\boldsymbol{u}_i = [\boldsymbol{u}_{i,0}, \dots, \boldsymbol{u}_{i,T-1}] \in \mathbb{R}^{m_i \times T}$. For simplicity, let $n := \sum_{i \in [M]} n_i$ and $m := \sum_{i \in [M]} m_i$. The state and control for all agents at time k are denoted by $\boldsymbol{x}_k = [\boldsymbol{x}_{1,k}, \dots, \boldsymbol{x}_{M,k}] \in \mathbb{R}^n$ and $\boldsymbol{u}_k = [\boldsymbol{u}_{1,k}, \dots, \boldsymbol{u}_{M,k}] \in \mathbb{R}^m$. We denote the entire state and control trajectory of all agents as $\boldsymbol{x} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_M] \in \mathbb{R}^{n \times (T+1)}$ and $\boldsymbol{u} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_M] \in \mathbb{R}^{m \times T}$. For each agent i and time instant k, we introduce safe variables $\tilde{\boldsymbol{x}}_{i,k} \in \mathbb{R}^{n_i}$, $\tilde{\boldsymbol{u}}_{i,k} \in \mathbb{R}^{m_i}$ that satisfy state and control constraints: $\boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \leq 0$ and $\boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \leq 0$. In addition, we introduce an augmented safe state variable $\tilde{\boldsymbol{x}}^a_{i,k} = [(\tilde{\boldsymbol{x}}^i_{j,k})_{j \in \mathcal{N}_i}]$, where $\tilde{\boldsymbol{x}}^i_{j,k} \in \mathbb{R}^{n_j}$ are safe copy variables in the neighborhood set $\mathcal{N}_i$. For simplicity, let $\tilde{n}_i = \sum_{j \in \mathcal{N}_i} n_j$, and denote the augmented pair-wise state constraint by $\boldsymbol{h}^a_i : \mathbb{R}^{\tilde{n}_i} \to \mathbb{R}$.

Specifically, the safe variables $\tilde{\boldsymbol{x}}_{i,k}$, $\tilde{\boldsymbol{u}}_{i,k}$ are projections of the unconstrained state $\boldsymbol{x}_{i,k}$ and control $\boldsymbol{u}_{i,k}$ onto the feasible safe region. To optimize consensus between these safe variables and the unconstrained state and control variables, we impose consensus constraints as $\tilde{\boldsymbol{x}}^a_{i,k} = \boldsymbol{z}^a_{i,k}$, $\tilde{\boldsymbol{x}}_{i,k} = \boldsymbol{x}_{i,k}$,

and $\tilde{\boldsymbol{u}}_{i,k} = \boldsymbol{u}_{i,k}$, where $\boldsymbol{z}^a_{i,k}$ is a global augmented variable. Consequently, the distributed optimal control is formulated as follows.

Problem 2 (Distributed optimal control problem)

$$\min_{\boldsymbol{u}} \sum_{i=1}^{M} J_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \sum_{i=1}^{M} \left[\Phi_i(\boldsymbol{x}_{i,T}) + \sum_{k=0}^{T-1} l_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}) \right] \tag{6}$$

$$\begin{aligned}
\textit{subject to} \quad & \boldsymbol{x}_{i,k+1} = \boldsymbol{F}_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}), && \forall i \in [M], \forall k = 0, \dots, T-1 && (7)\\
& \boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \leq 0, && \forall i \in [M], \forall k = 0, \dots, T-1, && (8)\\
& \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \leq 0, && \forall i \in [M], \forall k = 0, \dots, T && (9)\\
& \boldsymbol{h}^a_i(\tilde{\boldsymbol{x}}^a_{i,k}) \leq 0, && \forall i \in [M], \forall k = 0, \dots, T, && (10)\\
& \tilde{\boldsymbol{x}}^a_{i,k} = \boldsymbol{z}^a_{i,k}, \;\; \boldsymbol{x}_{i,k} = \tilde{\boldsymbol{x}}_{i,k}, && \forall i \in [M], \forall k = 0, \dots, T, && (11)\\
& \boldsymbol{u}_{i,k} = \tilde{\boldsymbol{u}}_{i,k}, && \forall i \in [M], \forall k = 0, \dots, T-1, && (12)
\end{aligned}$$

ADMM-DDP (Saravanos et al., 2023) is an efficient method to solve Problem 2, where the state and control constraints in (8)-(10) are handled in the safe update block, and the consensus constraints in (11)-(12) with dynamics constraints in (8) are incorporated in the DDP block. See Appendix D.2 for the derivation of the update rules of the primal, safe, global, and dual variables.

2.2 Model Predictive Distributed Differential Dynamic Programming

Although ADMM-DDP has a parallelizable architecture with tractable computational complexity, the neighborhood set $\mathcal{N}_i$ and the neighbor-of set $\mathcal{P}_i$ for each agent $i \in [M]$ are assumed to be fixed throughout the horizon, which may fail to capture the true neighborhood set if the locality structure of the multi-agent network evolves over time. To address this problem, we introduce Model Predictive Distributed Differential Dynamic Programming (MPD-DDP), a distributed nonlinear MPC method that extends ADMM-DDP. In MPD-DDP, we define a time-varying neighborhood set and neighbor-of set denoted by $\mathcal{N}_i^{(o)}$ and $\mathcal{P}_i^{(o)}$, which are recomputed at each replanning step (o). In particular, MPD-DDP solves Problem 2 in a receding horizon manner with neighborhood set reconstruction at each replanning step. MPD-DDP algorithm is described in Algorithm 1 in Appendix E. The effect of dynamic neighborhood set reconstruction in a time-varying problem structure can be found in Appendix I.6.

3 Multi-Agent Imitation Learning

The solve times of ADMM-DDP and MPD-DDP depend on the number of ADMM iterations. In a constrained environment with an increasing number of agents, ADMM-DDP and MPD-DDP often require a large number of ADMM iterations to converge, failing to meet the real time budget. To address this problem, we propose Multi-agent Imitation Learning via Distributed Model Predictive Control (MILD-MPC), a scalable and constraint-aware framework that learns to imitate a distributed MPC expert (MPD-DDP), which is also employed as a backup policy for deployment.

3.1 Multi-Agent Imitation Learning Framework

Conventionally, warm-starting the optimization process using time-shifted trajectories from the previous optimization step is a widely-used technique to accelerate optimization in real-time MPC (Diehl et al., 2002). In a similar vein, we warm-start DDP and ADMM (Toyoda & Tanaka, 2021) using the time-shifted trajectories of primal, dual, and global variables in MPD-DDP. However, time-shifted trajectories of dual variables are no longer valid as the neighborhood set changes from the previous replanning step, which hinders the effect of warm-starting ADMM.

To mitigate this issue, we frame the problem of accelerating the distributed MPC as a multi-agent imitation learning problem. The idea is that if the learned policy's trajectory does not violate the constraints within a certain time horizon and threshold, then the ADMM-based iterative optimization can be skipped. The overview of the proposed framework is illustrated in Fig 1.

Learning formulation. Let $\mathcal{X}, \mathcal{Y}$ be the problem context space and the control trajectory domain. The dataset is denoted as $\mathcal{T} = \{\{(\hat{x}_i^{(j,l)}, \hat{y}_i^{(j,l)})_{l=1}^{N^{\text{plan}}}\}_{i=1}^{M}\}_{j=1}^{N^{\text{prob}}}$, where N^{prob} is the number of problem instances, and N^{plan} is the number of replanning steps in an MPC loop. The terms $\hat{x}_i^{(j,l)} \in \mathbb{R}^{d^{\text{feat}}} \in \mathcal{X}$ and $\hat{y}_i^{(j,l)} \in \mathbb{R}^{m_i \times T} \in \mathcal{Y}$ denote the problem context and control trajectory solutions of agent i at replanning step l in problem j. We denote the learned policy by $\hat{y}_\theta : \mathcal{X} \to \mathcal{Y}$ parameterized by $\theta \in \Theta$, where Θ is the parameter space. In addition, let $\boldsymbol{x}_i^{(j,l)}$, $\boldsymbol{x}_{\theta,i}^{(j,l)}$ denote the state trajectory generated by propagating dynamics $\boldsymbol{F}_i$ using the expert and the learned control trajectories $\hat{y}_i^{(j,l)}$, $\hat{y}_\theta(\hat{x}_i^{(j,l)})$ of agent i. We denote the number of control constraints for agent i by N_i^{cc}, and the number of per-agent state constraints by N_i^g. Let $\tilde{\boldsymbol{b}}_i : \mathbb{R}^{m_i \times T} \to \mathbb{R}^{N_i^{cc} \times T}, \forall i \in [M]$, $\tilde{\boldsymbol{g}}_i : \mathbb{R}^{n_i \times (T+1)} \to \mathbb{R}^{N_i^g \times (T+1)}, \forall i \in [M]$, $\tilde{\boldsymbol{h}}_{i,o} : \mathbb{R}^{n_i \times (T+1)} \times \mathbb{R}^{n_k \times (T+1)} \to \mathbb{R}^{T+1}, \forall i \in [M], \forall o : i < o \leq M$ be the control trajectory constraint, per-agent state trajectory constraint, and inter-agent state trajectory constraint functions, which are all continuously differentiable.

The learning task is to approximate $\hat{y}_i^{(j,l)}$ using a learned policy $\hat{y}_\theta$ parameterized by θ, across agent i, replanning step l, and problem j. The parameters θ of the learned policy are learned by minimizing the following weighted loss function $L(\theta, \mathcal{X}, \mathcal{Y})$:

$$L(\theta, \mathcal{X}, \mathcal{Y}) \coloneqq \frac{1}{N^{\text{prob}} \cdot M} \sum_{j=1}^{N^{\text{prob}}} \sum_{i=1}^{M} \sum_{l=1}^{N^{\text{plan}}} \left[L_{i,j,l}^u + w^x L_{i,j,l}^x + w^b L_{i,j,l}^b + w^g L_{i,j,l}^g + w^h L_{i,j,l}^h \right], \quad (13)$$

where $w^b, w^x, w^g, w^h \geq 0$ are the weight coefficients for each individual loss function. For brevity, we omit the input notation $(\theta, \mathcal{X}, \mathcal{Y})$ in each loss function. The terms $L_{i,j,l}^u, L_{i,j,l}^x, L_{i,j,l}^b, L_{i,j,l}^g$, and $L_{i,j,l,k}^h$ are defined as follows:

$$L_{i,j,l}^u \coloneqq \|\hat{y}_i^{(j,l)} - \hat{y}_\theta(\hat{x}_i^{(j,l)})\|_2^2 / m_i, \qquad L_{i,j,l}^x \coloneqq \|\boldsymbol{x}_i^{(j,l)} - \boldsymbol{x}_{\theta,i}^{(j,l)}\|_2^2 / n_i, \quad (14)$$

$$L_{i,j,l}^b \coloneqq \| \max\left(\boldsymbol{0}, \tilde{\boldsymbol{b}}_i(\hat{y}_\theta(\hat{x}_i^{(j,l)}))\right)\|_2^2 / N_i^{cc}, \qquad L_{i,j,l}^g \coloneqq \| \max\left(\boldsymbol{0}, \tilde{\boldsymbol{g}}_i(\boldsymbol{x}_{\theta,i}^{(j,l)})\right)\|_2^2 / N_i^g, \quad (15)$$

$$L_{i,j,l}^h \coloneqq \frac{1}{M} \sum_{o=1}^{M} \left[\|\mathcal{I}_{i \neq k} \cdot \max\left(\boldsymbol{0}, \tilde{\boldsymbol{h}}_{i,o}(\boldsymbol{x}_{\theta,i}^{(j,l)}, \boldsymbol{x}_{\theta,o}^{(j,l)})\right)\|_2^2 \right]. \quad (16)$$

Notice that our state matching loss $L_{i,j,l}^x$ requires propagating dynamics $\boldsymbol{F}_i$. Hence, our learning formulation relates to imitation learning with the known-transition setting under a deterministic expert policy (Rajaraman et al., 2020), which overcomes the quadratic error compounding barrier of behavior cloning. The final loss $L(\theta, \mathcal{X}, \mathcal{Y})$ is minimized using the Adam optimizer (Kingma & Ba, 2014) with a cosine decay learning rate scheduler (Loshchilov & Hutter, 2016).

Permutation invariant Transformer architecture. Problem contexts in multi-agent systems may vary in the number of agents M, the number of neighbors $|\mathcal{N}_i|$ for each agent i, and the number of constraints. Additionally, the ordering of the agents, neighbors, and constraints in the problem context can be arbitrary. Hence, we consider the following principles: The model should be (i) permutation invariant with respect to the ordering of the input features in the problem context for each agent, (ii) permutation equivariant with respect to the ordering of the agents. Otherwise, the ordering of input features or agents can be a source of model vulnerability (Chen et al., 2025; Huang et al., 2017). To adhere to these principles, we employ a permutation invariant Transformer architecture based on Set Transformer (Lee et al., 2019) to parameterize the model $\hat{y}_\theta$. Consequently, we suggest the following proposition, extending the result in Lee et al. (2019).

Proposition 1 *If there exist continuous optimal control policies $\pi_i^{policy} : \mathbb{R}^{d^{feat}} \to \mathbb{R}^{m_i \times T}$ for all $i \in [M]$ in the distributed optimal control problem in Problem 2, the permutation invariant Transformer model is a universal approximator of π_i^{policy} for all $i \in [M]$.*

Proof *See Appendix H.*

Architecture details are introduced in Appendix G. The overall model architecture is illustrated in Fig. 4. Details on the input features can be found in Appendix F.

3.2 Multi-Agent Predictive Safety Filter

In the deployment phase, the learned policy predicts a control trajectory $\hat{y}_\theta(\hat{x}_i^{(j,l)}) \in \mathcal{Y}$ for each agent i in problem j at replanning step l. Since $\hat{y}_\theta$ is an approximation of the expert, its constraint satisfaction is not guaranteed. Hence, we devise a relaxed safety module by formulating a multi-agent predictive safety filter, extending a predictive safety filter (Wabersich & Zeilinger, 2021).

Formulation. Let T^{safe} be the prediction horizon for the safety filter. We denote the learned policy's trajectory as $\boldsymbol{u}_{\theta,i,0:T^{\text{safe}}-1} := \hat{y}_\theta(\hat{x}_i)$. Let $\boldsymbol{x}_{\theta,i,0:T^{\text{safe}}}$ be the state trajectory obtained by propagating dynamics using $\boldsymbol{u}_{\theta,i,0:T^{\text{safe}}-1}$ given the initial state $\boldsymbol{x}_{\theta,i,0}$. For simplicity, we denote $\boldsymbol{u}_{\theta,i} := \boldsymbol{u}_{\theta,i,0:T^{\text{safe}}-1}$, $\boldsymbol{x}_{\theta,i} := \boldsymbol{x}_{\theta,i,0:T^{\text{safe}}}$, $\tilde{\boldsymbol{u}}_i := \tilde{\boldsymbol{u}}_{i,0:T^{\text{safe}}-1}$, $\tilde{\boldsymbol{x}}_i := \tilde{\boldsymbol{x}}_{i,0:T^{\text{safe}}}$. Similarly, let $\boldsymbol{u}_\theta := [(\boldsymbol{u}_{\theta,i})_{i=1}^M]$, $\boldsymbol{x}_\theta := [(\boldsymbol{x}_{\theta,i})_{i=1}^M]$, $\tilde{\boldsymbol{u}} := [(\tilde{\boldsymbol{u}}_i)_{i=1}^M]$, $\tilde{\boldsymbol{x}} := [(\tilde{\boldsymbol{x}}_i)_{i=1}^M]$. Given the predicted control trajectory $\boldsymbol{u}_\theta$ and its corresponding state trajectory $\boldsymbol{x}_\theta$, we solve the following problem.

Problem 3 (Multi-agent predictive safety filter problem)

$$\min_{\tilde{\boldsymbol{x}},\tilde{\boldsymbol{u}}} \sum_{i=1}^{M} \left[\Phi_i^{safe}(\tilde{\boldsymbol{x}}_{i,T}) + \sum_{t=0}^{T^{safe}-1} [l_i^{safe}(\boldsymbol{u}_{\theta,i,t}, \tilde{\boldsymbol{u}}_{i,t})]\right] \tag{17}$$

$$\begin{aligned}
\textit{subject to} \quad & \tilde{\boldsymbol{x}}_{i,k+1} = \boldsymbol{F}_i(\tilde{\boldsymbol{x}}_{i,k}, \tilde{\boldsymbol{u}}_{i,k}), && \forall i \in [M], \forall k = 0, \dots, T^{safe}-1, && (18)\\
& \boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \le 0, && \forall i \in [M], \forall k = 0, \dots, T^{safe}-1, && (19)\\
& \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \le 0, && \forall i \in [M], \forall k = 0, \dots, T^{safe}, && (20)\\
& \tilde{\boldsymbol{x}}_{i,0} = \boldsymbol{x}_{\theta,i,0}, && \forall i \in [M], && (21)
\end{aligned}$$

where $\Phi_i^{\text{safe}} : \mathbb{R}^{n_i} \to \mathbb{R}$ is the terminal cost function and $l_i^{\text{safe}} : \mathbb{R}^{m_i} \times \mathbb{R}^{m_i} \to \mathbb{R}$ is the stage cost function for the safety filter. The main difference of Problem 3 from the formulations in Wabersich & Zeilinger (2021); Oshin et al. (2025) is that we introduce the terminal cost function Φ_i^{safe} in the objective function, which provides flexibility to steer the learned policy's trajectory toward the target state. Furthermore, Problem 3 can be decomposed into M subproblems, which can be solved efficiently in parallel. We do not impose inter-agent state constraints (10) in Problem 3 as a relaxation of the original set of constraints. The motivation for this relaxation is that memory and computation costs can be reduced if the solution to the relaxed problem is already feasible for the original set of constraints, while maintaining optimality (Dantzig et al., 1954; Beale, 1959).

3.3 Verifier Gating Mechanism

While the feasible solution to Problem 3 satisfies per-agent constraints in (19)-(20), we still need to ensure inter-agent constraint satisfaction. In the verifier stage, we validate the applicability of the candidate trajectories $\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}$ through three sequential gates: (i) control constraint gate, (ii) state constraint gate, and (iii) state constraint residual gate.

The candidate control trajectory passes the control constraint gate if the following holds: $\sup_{i\in[M],k:0\le k<T^{\text{ver, cnst}}} \boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \le \tau_b$, where $\boldsymbol{b}_i$ is the control constraint of agent i and τ_b is the threshold value for the control constraint gate. Similarly, the candidate state trajectory is validated by the state constraint gate if the following conditions are true:

$$\sup_{i\in[M],\ k:0\le k\le T^{\text{ver, cnst}}} \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \le \tau_g, \qquad \sup_{i,j:1\le i<j\le M,\ k:0\le k\le T^{\text{ver, cnst}}} \boldsymbol{h}_{i,j}(\tilde{\boldsymbol{x}}_{i,k}, \tilde{\boldsymbol{x}}_{j,k}) \le \tau_h,$$

where $\boldsymbol{g}_i$ is the per-agent state constraint, $\boldsymbol{h}_i$ is the inter-agent state constraint, τ_g and τ_h are the threshold values for the state constraint gate. We define the state constraint residuals as $\bar{\boldsymbol{g}}^+ := (\sum_{i=1}^{M}\sum_{k=0}^{T^{\text{ver, res}}} \max(0, \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}))^2)^{1/2}$, $\bar{\boldsymbol{h}}^+ := (\sum_{i=1}^{M}\sum_{j=i+1}^{M}\sum_{k=0}^{T^{\text{ver, res}}} \max(0, \boldsymbol{h}_{i,j}(\tilde{\boldsymbol{x}}_{i,k}, \tilde{\boldsymbol{x}}_{j,k}))^2)^{1/2}$.

Finally, the validation condition of the state constraint residual gate is expressed as $\|[\bar{\boldsymbol{g}}^+, \bar{\boldsymbol{h}}^+]\|_2 < \epsilon$, where ϵ is the threshold value and $T^{\text{ver, res}}$ is the look-ahead horizon for the state constraint residual gate. If the candidate trajectory is validated through all three gates, then we apply the candidate control trajectory $\tilde{\boldsymbol{u}}$ to the system; otherwise, we use MPD-DDP.

4 EXPERIMENTS

In this section, we will investigate the main questions for the proposed method regarding: (i) the comparison with the centralized baseline, (ii) the generalizability in small-to-large problem transfer at deployment, (iii) the robustness in agent failure scenarios, and (iv) the effect of neural network architectures in the proposed framework. Finally, we test the physical applicability of the proposed method on the Robotarium swarm hardware platform in real time MPC. In our experiments, we use offboard computation on a single machine. Details on dynamics, constraints, cost functions, algorithm configurations, and trajectory visualizations can be found in Appendix I.

4.1 SIMULATIONS

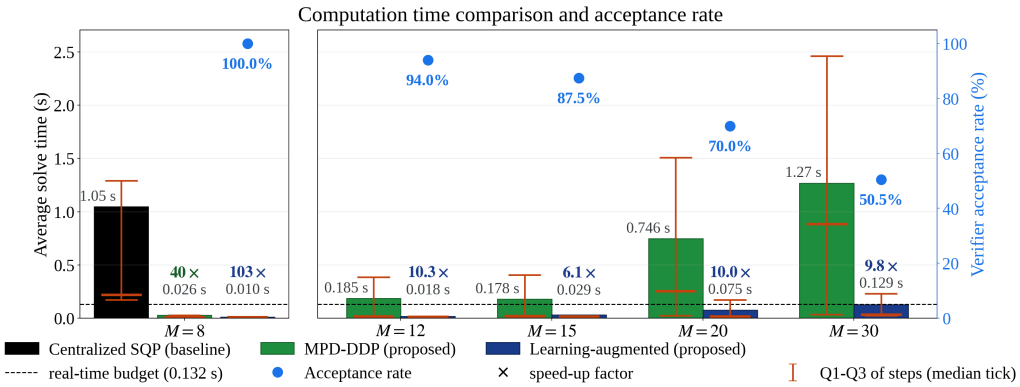


Figure 2: Obstacle-field navigation tasks with two moving obstacles on a multi-agent unicycle system: the learning-augmented policy's average solve time remains within the time budget up to 30 agents, achieving a $\mathbf{103}\times$ speedup over centralized Sequential Quadratic Programming (SQP) baseline at 8-agent and around $\mathbf{10}\times$ speedup over MPD-DDP at 20 and 30-agent settings. *Left*: Centralized SQP baseline, MPD-DDP, and the learning-augmented policy at 8-agent setting. Speedup values are relative to SQP. *Right*: Transfer of 8-agent policy to $(12, 15, 20, 30)$ agent problems. Speedup values are relative to MPD-DDP. Solve time is averaged over $N^{\text{plan}} = 200$ steps for each setting, with a time budget of 0.132 s at $dt = 0.033$ s. The verifier acceptance rate (•) is the fraction of steps at which the verifier validates the learned policy's prediction. Invalidated steps fall back to MPD-DDP. The interquartile range (IQR) in a red error bar represents the solve time quartile range from Q1 to Q3 across replanning steps.

How does the proposed method compare to the baseline? We test the efficacy of the proposed method in comparison with the centralized MPC baseline. We train the policy using expert demonstrations generated by MPD-DDP on an 8-agent unicycle system with two moving obstacles and randomized initial and target positions. For the baseline solver, we employ a multi-shooting Sequential Quadratic Programming (SQP) using a sequential $l1$-QP model (Han, 1977; Fletcher, 2013) and OSQP solver (Stellato et al., 2020). In the left plot of Fig. 2, the learning-augmented policy shows $\mathbf{103}\times$ speedup with $\mathbf{100}\%$ verifier acceptance rate, and MPD-DDP solves $\mathbf{40}\times$ faster compared to the baseline. All three algorithms complete the task safely, *i.e.*, all agents reached the target position within 0.15 m without any collisions.

In Table 1, we also compare the proposed method with the centralized SQP in a 4-drone swapping task in a forest-like environment with 17 dense obstacles in real-time MPC. Specifically, we use the Gazebo simulator (Koenig & Howard, 2004) in PX4 Software-In-The-Loop (SITL) enabled with Extended Kalman Filter (EKF) state estimation and physical sensor emulation. In the nominal scenario, the learning-augmented policy and MPD-DDP achieve $\mathbf{36.6}\times, \mathbf{25.86}\times$ speedup over the baseline, safely completing the task in every trial, where all agents reached their target positions within 0.2 m without any collisions. The centralized SQP baseline does not complete the task at all, failing to meet the 2-minute time limit.

Does the proposed method generalize in small-to-large problem transfer? We verify the generalizability of the proposed method in small-to-large problem transfer. We deploy the policy trained on 8-agent problems to the $(12, 15, 20, 30)$ agent unicycle systems, which involves two moving obstacles. In the right plot of Fig. 2, the learning-augmented policy deployed on the $(12, 15, 20, 30)$-agent settings shows $(94, 87.5, 70, 50.5)\%$ acceptance rates, which is the fraction of steps that are validated by the verifier. In this task, we also demonstrate that the learning-augmented approach achieves a $(\mathbf{10.3}, \mathbf{6.1}, \mathbf{10}, \mathbf{9.8})\times$ speedup in $(12, 15, 20, 30)$-agent settings over MPD-DDP. In this experiment, we omit the centralized SQP baseline results, since their total run times exceed the 20-minute time limit in every setting. The number of neighbors $|\mathcal{N}_i|$ for each agent i is assigned $|\mathcal{N}_i| = 2$ for $(12, 15)$-agent settings and $|\mathcal{N}_i| = 4$ for $(20, 30)$-agent settings, which demonstrates the generalizability of the permutation invariant Transformer model in the number of neighbors.

We observe that warm-starting the backup policy (MPD-DDP) with the learned trajectory helps reduce the maximum ADMM iterations required for collision-free results. Accordingly, the maximum number of ADMM iterations is $(5, 30, 20, 45, 60)$ for MPD-DDP, and uniformly 5 for the learning-augmented policy, in $(8, 12, 15, 20, 30)$-agent settings. As a result, MPD-DDP shows a larger interquartile range in Fig. 2. The verifier acceptance rate decreases as the number of agents increases at deployment, *i.e.*, from 94% in 12-agent to 50.5% in 30-agent settings. However, the rejected steps are routed to the backup policy (MPD-DDP), completing the task safely; all agents reached the target position within 0.15 m without any collisions. It is noteworthy that $(20, 30)$ agent problems are out-of-distribution compared to the dataset used for training the policy. All results for each method in Fig. 2 complete the task collision-free. Each configuration is repeated five times to reduce variance on the average timing results. To provide reproducible results in a controlled environment, this experiment assumes a zero-delay setup.

Scenario	Method	Safe TC ↑	Solve time (s) ↓	Speedup ↑
4-drone swap	Learning-aug (proposed)	**5/5**	**0.072**	**36.6×**
	MPD-DDP (proposed)	5/5	0.102	25.86×
	Centralized SQP (baseline)	0/5	2.638	—
4-drone swap + 1-agent failure	Learning-aug (proposed)	**5/5**	**0.076**	**22.6×**
	MPD-DDP (proposed)	5/5	0.452	3.81×
	Centralized SQP (baseline)	3/5	1.721	—

Table 1: The learning-augmented policy and MPD-DDP show $(\mathbf{36.6}\times, \mathbf{25.86}\times)$ speedup over the centralized SQP baseline in the nominal 4-drone swapping task with 17 dense obstacles, safely completing the task in all 5 trials. In the agent failure scenario, the learning-augmented policy remains robust and efficient, being $\mathbf{22.6}\times$ faster than the baseline while satisfying safe task completion (TC) in all 5 trials, which counts only operating agents.

Model	Ordering	Safe TC ↑ (/99)
Permutation invariant Transformer	nominal	57
	reversed	57
Transformer (Vaswani et al., 2017)	nominal	58
	reversed	0
Deep Sets (Zaheer et al., 2017)	nominal	0
	reversed	0
RNN (Cho et al., 2014)	nominal	0
	reversed	0

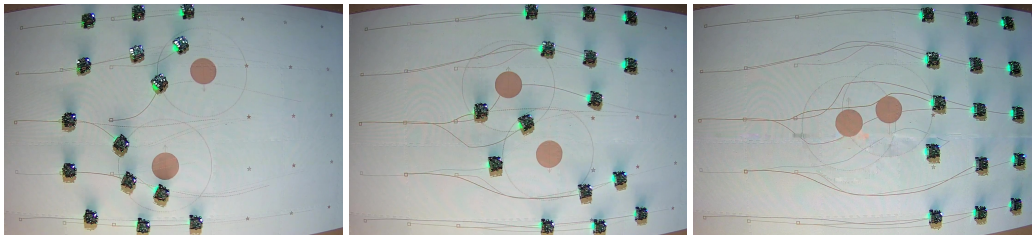
Table 2: On-policy evaluation results of the learned policy without a predictive safety filter or backup policy in obstacle-field navigation tasks with 8 unicycle agents and 6 obstacles in randomized positions and sizes.

Is the proposed method robust against agent failure scenarios? Agent failure scenarios are among the most common cases of contingency in multi-agent systems, which need to be thoroughly tested in advance of real world deployment. We test our methods in comparison with the centralized SQP on a 4-drone swapping task with 17 tree-like obstacles in real-time MPC with physical sensor emulation and EKF state estimation. In Table 1, we observe that the learning-augmented policy remains robust and efficient in the agent failure scenario, demonstrating $\mathbf{22.6}\times$ faster solve time than the centralized SQP baseline. In the agent failure scenario, MPD-DDP shows a large control consensus residual between the planned control and the safe control, *i.e.*, $\|\boldsymbol{u} - \tilde{\boldsymbol{u}}\|^2$ in (12), causing a higher number of ADMM iterations and $4.43\times$ slower solve time compared to the nominal scenario. Nonetheless, MPD-DDP performs $3.81\times$ faster than the centralized SQP. Consequently, the learning-augmented policy achieves $\mathbf{5.95}\times$ speedup against the MPD-DDP in the agent failure scenario. In this experiment, safe task completion (TC) is the condition where all agents reached the target positions within 0.2 m collision-free.

What is the effect of the neural network architecture in solving multi-agent control problems? To understand the contribution of the learned policy parameterized by the neural network in the proposed framework, we investigate the on-policy performance of standard neural network architectures in solving the multi-agent control problem. In this experiment, the multi-agent predictive

safety filter and the backup policy are not deployed. The number of parameters in all models is less than 1 million. In Table 2, the permutation invariant Transformer remains robust to input token ordering. The vanilla Transformer (Vaswani et al., 2017) matches the performance of the permutation invariant Transformer in the nominal input ordering, *i.e.*, $(58, 57)$ safe task completions (TC) out of 99 test problems, but collapses to zero under input token reversal, while Deep Sets (Zaheer et al., 2017) and RNN (Cho et al., 2014) fail to solve any problems regardless of ordering. Safe TC counts collision-free instances with all agents reaching their target positions within 0.15 m. Reversed ordering changes the input block order from (self, obstacle, neighbor) to (neighbor, obstacle, self) and reverses the order of the input tokens within each block.

4.2 Hardware Experiment



(a) 8-to-15 agent transfer at 4 s (b) 8-to-15 agent transfer at 9 s (c) 8-to-15 agent transfer at 36 s

Figure 3: Robotarium swarm robotics hardware experiments: The policy trained with 8 agents is deployed to 15-agent setting. Two red circles in the center represents moving obstacle. All agents complete the task within 36 s in this experiment without any collisions.

In this section, we demonstrate that the proposed method generalizes in small-to-large problem transfer on the Robotarium swarm robotics testbed (Pickem et al., 2017; Wilson et al., 2020) and runs on the MPC loop asynchronously in real time, where computation delay affects the results. Fig. 3 illustrates the trajectories of the learning-augmented policy in the 8-to-15 agent problem transfer experiment on the Robotarium platform. Across all three repetitions, the proposed method successfully finished the task, *i.e.*, all agents reached their target positions within 0.15 m with zero collisions. In Robotarium, the (average, median, maximum) solve times are recorded $(0.152, 0.127, 0.447)$ s, and the (average, maximum) numbers of ADMM iterations are $(1.25, 3)$. The safety verifier accepted 87.6% of the learned trajectories out of 418 solves, which closely align with the result in Fig. 2.

5 Conclusions

We present a new framework, Multi-agent Imitation Learning via Distributed Model Predictive Control (MILD-MPC), where the policy is learned using demonstrations from a distributed MPC expert, which is also employed as a backup policy at deployment. In MILD-MPC, we introduce (i) Model Predictive Distributed Differential Dynamic Programming (MPD-DDP), a new distributed nonlinear MPC method that supports time-varying neighborhood structures, (ii) a multi-agent predictive safety filter formulation to provide a relaxed constraint handling layer for the learned policy. Through simulations and hardware experiments, MILD-MPC is tested on a series of tasks with dense active constraints in multi-agent unicycle and quadrotor systems. Specifically, the proposed method is verified on the generalizability in the problem size transfer, robustness to agent failure scenarios, and the effect of the neural network architecture. To our knowledge, MILD-MPC is the first unified framework that incorporates multi-agent imitation learning with distributed nonlinear MPC.

In the future, improving the scalability of the distributed optimizer can be a straightforward way to boost the performance of the proposed framework. Furthermore, we believe that self-supervised learning and control barrier function extensions could provide evidence for improving the scalability of the multi-agent controller by reducing the dependency on the expert. Finally, the distributed MPC method (MPD-DDP) requires tuning for a reliable data generation or deployment. In this regard, optimizing the distributed MPC as a differentiable layer (Amos et al., 2018; Oshin et al., 2023) in the learning framework could lead to more flexible and robust multi-agent control.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs, cleaning or reformatting dataset. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for identifying relevant literature, editing the draft to improve readability, searching for information, summarizing or analyzing existing literature, creating or editing software code, creating or modifying scientific figures or images. We have reviewed all AI-assisted work. For example, we have reviewed LLM-generated codes, mathematical expressions, and figures for correctness. Also, LLM-assisted literature search or summary was manually verified by the author. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Code to reproduce the results in Section 4.1 is available at https://osf.io/h7gwj/files/osfstorage?view_only=c84361a70ee444eabf69c82f19b343cb.

REFERENCES

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)*, pp. 3420–3431. Ieee, 2019.

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

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

EML Beale. On quadratic proramming. *Naval Research Logistics Quarterly*, 6(3):227–243, 1959.

Eduardo Camponogara, Dong Jia, Bruce H Krogh, and Sarosh Talukdar. Distributed model predictive control. *IEEE control systems magazine*, 22(1):44–52, 2002.

Liang Chen, Li Shen, Yang Deng, Xiaoyan Zhao, Bin Liang, and Kam-Fai Wong. Pearl: Towards permutation-resilient llms. In *International Conference on Learning Representations*, volume 2025, pp. 101083–101102, 2025.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

Moritz Diehl, H Georg Bock, Johannes P Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok, and Alejandro Ribeiro. Synthesizing decentralized controllers with graph neural networks and imitation learning. *IEEE Transactions on Signal Processing*, 70:1932–1946, 2022.

R Haftka and J STARNES, JR. Applications of a quadratic extended interior penalty function for structural optimization. In *16th Structural Dynamics, and Materials Conference*, pp. 764, 1976.

Trevor Halsted, Ola Shorinwa, Javier Yu, and Mac Schwager. A survey of distributed optimization methods for multi-robot systems. *arXiv preprint arXiv:2103.12840*, 2021.

Shih-Ping Han. A globally convergent method for nonlinear programming. *Journal of optimization theory and applications*, 22(3):297–309, 1977.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Michael Hertneck, Johannes Köhler, Sebastian Trimpe, and Frank Allgöwer. Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3):543–548, 2018.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Zhenmin Huang, Shaojie Shen, and Jun Ma. Decentralized ilqr for cooperative trajectory planning of connected autonomous vehicles via dual consensus admm. *IEEE Transactions on Intelligent Transportation Systems*, 24(11):12754–12766, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pp. 2149–2154. Ieee, 2004.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pp. 1–9. PMLR, 2013.

Haichao Liu, Zhenmin Huang, Zicheng Zhu, Yulin Li, Shaojie Shen, and Jun Ma. Improved consensus admm for cooperative motion planning of large-scale connected autonomous vehicles with limited communication. *IEEE Transactions on Intelligent Vehicles*, 9(11):7222–7238, 2024.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Carlos E Luis, Marijan Vukosavljev, and Angela P Schoellig. Online trajectory generation with distributed model predictive control for multi-robot motion planning. *IEEE Robotics and Automation Letters*, 5(2):604–611, 2020.

Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22(22):1–24, 2011.

José M Maestre, Rudy R Negenborn, et al. *Distributed model predictive control made easy*, volume 69. Springer, 2014.

Lorenz Meier and The PX4 Contributors. PX4 Autopilot. URL https://px4.io.

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2): 1–179, 2018.

Alex Oshin, Hassan Almubarak, and Evangelos A Theodorou. Differentiable robust model predictive control. *arXiv preprint arXiv:2308.08426*, 2023.

Alex Oshin, Rahul Vodeb Ghosh, Augustinos D Saravanos, and Evangelos A Theodorou. Deep flexqp: Accelerated nonlinear programming via deep unfolding. *arXiv preprint arXiv:2512.01565*, 2025.

Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1699–1706. IEEE, 2017.

Nived Rajaraman, Lin Yang, Jiantao Jiao, and Kannan Ramchandran. Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924, 2020.

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.

R Tyrrell Rockafellar. A dual approach to solving nonlinear programming problems by unconstrained optimization. *Mathematical programming*, 5(1):354–373, 1973.

Augustinos D Saravanos, Yuichiro Aoyama, Hongchang Zhu, and Evangelos A Theodorou. Distributed differential dynamic programming architectures for large-scale multiagent control. *IEEE Transactions on Robotics*, 2023.

Augustinos D. Saravanos, Isin M. Balci, Efstathios Bakolas, and Evangelos A. Theodorou. Distributed model predictive covariance steering. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5740–5747, 2024. doi: 10.1109/IROS58592.2024.10802104.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12 (4):637–672, 2020.

Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro. Learning decentralized controllers for robot swarms with graph neural networks. In *Conference on robot learning*, pp. 671–682. PMLR, 2020.

Mitsuru Toyoda and Mirai Tanaka. An analysis of hot-started admm for linear mpc. *IET Control Theory & Applications*, 15(15):1999–2016, 2021.

Hiroyasu Tsukamoto and Soon-Jo Chung. Learning-based robust motion planning with guaranteed stability: A contraction theory approach. *IEEE Robotics and Automation Letters*, 6(4):6164–6171, 2021.

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

Kim Peter Wabersich and Melanie N Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597, 2021.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Sean Wilson, Paul Glotfelter, Li Wang, Siddharth Mayya, Gennaro Notomista, Mark Mote, and Magnus Egerstedt. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 40(1):26–44, 2020.

Sidney Yakowitz and Brian Rutherford. Computational aspects of discrete-time optimal control. *Applied Mathematics and Computation*, 15(1):29–45, 1984.

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.

Libin Zhang, Ping Wang, and Yunze Cai. Learning-based distributed mpc for nonconvex consensus optimization with collision constraints. *IEEE Transactions on Automation Science and Engineering*, 22:23784–23797, 2025a.

Songyuan Zhang, Oswin So, Kunal Garg, and Chuchu Fan. Gcbf+: A neural graph control barrier function framework for distributed safe multiagent control. *IEEE Transactions on Robotics*, 41: 1533–1552, 2025b.

Ziyi Zhou, Pengyuan Shu, Ruize Cao, Yuntian Zhao, and Ye Zhao. Aclm: Admm-based distributed model predictive control for collaborative loco-manipulation. *arXiv preprint arXiv:2603.07095*, 2026.

A Appendix

A Differential dynamic programming

We consider a discrete time system governed by the nonlinear dynamics described as $\boldsymbol{x}_{k+1} = \boldsymbol{F}(\boldsymbol{x}_k, \boldsymbol{u}_k)$, where $\boldsymbol{x}_k \in \mathbb{R}^n$ is the state, $\boldsymbol{u}_k \in \mathbb{R}^m$ is the input at the time instant k. We define the state trajectory from time k to T as $\boldsymbol{x}_{k:T} \coloneqq [\boldsymbol{x}_k, \dots, \boldsymbol{x}_T] \in \mathbb{R}^{n\times(T-k+1)}$ and the control trajectory from time k to $T-1$ as $\boldsymbol{u}_{k:T-1} \coloneqq [\boldsymbol{u}_k, \dots, \boldsymbol{u}_{T-1}] \in \mathbb{R}^{m\times(T-k)}$. Let $J : \mathbb{R}^{n\times(T-k+1)} \times \mathbb{R}^{m\times(T-k)} \to \mathbb{R}$ be the cost function defined as $J(\boldsymbol{x}_{k:T}, \boldsymbol{u}_{k:T-1}) \coloneqq \Phi(\boldsymbol{x}_T) + \sum_{t=k}^{T-1} l(\boldsymbol{x}_t, \boldsymbol{u}_t)$, where $\Phi : \mathbb{R}^n \to \mathbb{R}$ is a terminal cost function and $l : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is a stage cost function. Our goal is to find a control trajectory that minimizes the cost function, formulated as $\boldsymbol{u}^*_{k:T-1} = \arg\min_{\boldsymbol{u}_{k:T-1}} J(\boldsymbol{x}_{k:T}, \boldsymbol{u}_{k:T-1})$. We define the value function $V : \mathbb{R}^n \to \mathbb{R}$ as $V(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_{k:T-1}} J(\boldsymbol{x}_{k:T}, \boldsymbol{u}_{k:T-1})$. Subsequently, we define the Q-function $Q : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ using the value function as $Q(\boldsymbol{x}_k, \boldsymbol{u}_k) = l(\boldsymbol{x}_k, \boldsymbol{u}_k) + V(\boldsymbol{x}_{k+1})$. Furthermore, the value function and the Q-function is related as $V(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k} Q(\boldsymbol{x}_k, \boldsymbol{u}_k)$. Given a boundary condition $V(\boldsymbol{x}_T) = \Phi(\boldsymbol{x}_t)$ and nominal trajectory $(\bar{\boldsymbol{x}}_{k:T}, \bar{\boldsymbol{u}}_{k:T-1})$, we perform a second-order approximation of the Q-function around the nominal trajectory. We refer readers to Appendix B for the derivation of the optimal control deviation $\delta\boldsymbol{u}_k^{*(i)}$ based on the second-order approximation of the Q-function and the value function. We iteratively update $\boldsymbol{u}_k^{(i+1)} \leftarrow \boldsymbol{u}_k^{(i)} + \delta\boldsymbol{u}_k^{*(i)}$ until the cost J converges, where the superscript $^{(i)}$ represents the i-th differential dynamic programming (DDP) iteration.

B Derivation of the optimal control deviation in DDP

Define the local state and control perturbation as $\delta\boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k$ and $\delta\boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k$. Formally,

$$Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k) \approx Q_0 + Q_{\boldsymbol{x}}\delta\boldsymbol{x}_k + Q_{\boldsymbol{u}}\delta\boldsymbol{u}_k + \frac{1}{2}\begin{bmatrix}\delta\boldsymbol{x}_k\\ \delta\boldsymbol{u}_k\end{bmatrix}^\top \begin{bmatrix}Q_{\boldsymbol{xx}} & Q_{\boldsymbol{xu}}\\ Q_{\boldsymbol{ux}} & Q_{\boldsymbol{uu}}\end{bmatrix}\begin{bmatrix}\delta\boldsymbol{x}_k\\ \delta\boldsymbol{u}_k\end{bmatrix}, \tag{22}$$

where $Q_0 = Q(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$. The coefficients in (22) are calculated as

$$\begin{aligned} &Q_{\boldsymbol{x}} = l_{\boldsymbol{x}} + \boldsymbol{F}_{\boldsymbol{x}}^\top V_{\boldsymbol{x}}^+, \qquad Q_{\boldsymbol{u}} = l_{\boldsymbol{u}} + \boldsymbol{F}_{\boldsymbol{u}}^\top V_{\boldsymbol{x}}^+, \qquad Q_{\boldsymbol{xx}} = l_{\boldsymbol{xx}} + \boldsymbol{F}_{\boldsymbol{x}}^\top V_{\boldsymbol{xx}}^+ \boldsymbol{F}_{\boldsymbol{x}} + V_{\boldsymbol{x}}^+ \cdot \boldsymbol{F}_{\boldsymbol{xx}},\\ &Q_{\boldsymbol{uu}} = l_{\boldsymbol{uu}} + \boldsymbol{F}_{\boldsymbol{u}}^\top V_{\boldsymbol{xx}}^+ \boldsymbol{F}_{\boldsymbol{u}} + V_{\boldsymbol{x}}^+ \cdot \boldsymbol{F}_{\boldsymbol{uu}}, \qquad Q_{\boldsymbol{ux}} = l_{\boldsymbol{ux}} + \boldsymbol{F}_{\boldsymbol{u}}^\top V_{\boldsymbol{xx}}^+ \boldsymbol{F}_{\boldsymbol{x}} + V_{\boldsymbol{x}}^+ \cdot \boldsymbol{F}_{\boldsymbol{ux}}. \end{aligned} \tag{23}$$

We find the optimal control perturbation $\delta\boldsymbol{u}_k^*$ by minimizing the quadratic approximation of the Q-function with respect to $\delta\boldsymbol{u}_k$: $\boldsymbol{0} = \frac{\partial Q(\delta\boldsymbol{x}_k, \delta\boldsymbol{u}_k)}{\partial(\delta\boldsymbol{u}_k)} = Q_{\boldsymbol{u}} + Q_{\boldsymbol{uu}}\delta\boldsymbol{u}_k + Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k$, which implies $\delta\boldsymbol{u}_k^* = -Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{u}} - Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{ux}}\delta\boldsymbol{x}_k$, where $-Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{u}} = \boldsymbol{k}_k \in \mathbb{R}^m$ is the open-loop gain vector and $-Q_{\boldsymbol{uu}}^{-1}Q_{\boldsymbol{ux}} = \boldsymbol{K}_k \in \mathbb{R}^{m\times n}$ is the feedback gain matrix. We consider the quadratic expansion of the value function around the nominal state $\bar{\boldsymbol{x}}_k$ as $V(\delta\boldsymbol{x}_k) \approx V_0 + V_{\boldsymbol{x}}\delta\boldsymbol{x}_k + \frac{1}{2}\delta\boldsymbol{x}_k^\top V_{\boldsymbol{xx}}\delta\boldsymbol{x}_k$, where $V_0 = V(\bar{\boldsymbol{x}}_k)$. Substituting $\delta\boldsymbol{u}_k^*$ into (22) and using the relationship between the Q-function and the value function yields the following.

$$\frac{\partial V(\delta\boldsymbol{x}_k)}{\partial(\delta\boldsymbol{x}_k)} = Q_{\boldsymbol{x}} - K_k^\top Q_{\boldsymbol{uu}}k_k, \qquad \frac{\partial^2 V(\delta\boldsymbol{x}_k)}{\partial(\delta\boldsymbol{x}_k)^2} = Q_{\boldsymbol{xx}} - K_k^\top Q_{\boldsymbol{uu}}K_k \tag{24}$$

We substitute (24) into (23) and backpropagate over time from $k = T$ to $k = 0$.

C NESTED DISTRIBUTED DDP

Problem 1 can be naturally decomposed into M subproblems, except for pair-wise state constraints in (5). To decompose pair-wise state constraints in (5), ND-DDP introduces augmented variables in the neighborhood level and imposes consensus constraints between copied variables and original variables. For simplicity, let $\tilde{n}_i \coloneqq \sum_{j\in\mathcal{N}_i} n_j$, $\tilde{m}_i \coloneqq \sum_{j\in\mathcal{N}_i} m_j$, $\tilde{n} \coloneqq \sum_{i\in[M]} \tilde{n}_i$, and $\tilde{m} \coloneqq \sum_{i\in[M]} \tilde{m}_i$. Formally, we define the augmented trajectories of all agents as $\boldsymbol{x}^a = [(\boldsymbol{x}_i^a)_{i=1}^M] \in \mathbb{R}^{\tilde{n}\times(T+1)}$, $\boldsymbol{u}^a = [(\boldsymbol{u}_i^a)_{i=1}^M] \in \mathbb{R}^{\tilde{m}\times T}$, where $\boldsymbol{x}_i^a = [(\boldsymbol{x}_{i,k}^a)_{k=0,\dots,T}] \in \mathbb{R}^{\tilde{n}_i\times(T+1)}$ and $\boldsymbol{u}_i^a = [(\boldsymbol{u}_{i,k}^a)_{k=0,\dots,T-1}] \in \mathbb{R}^{\tilde{m}_i\times T}$. The augmented state and control variables are defined as $\boldsymbol{x}_{i,k}^a = [(\boldsymbol{x}_{j,k}^i)_{j\in\mathcal{N}_i}]$ and $\boldsymbol{u}_{i,k}^a = [(\boldsymbol{u}_{j,k}^i)_{j\in\mathcal{N}_i}]$, where $\boldsymbol{x}_{j,k}^i \in \mathbb{R}^{n_j}$, $\boldsymbol{u}_{j,k}^i \in \mathbb{R}^{m_j}$ are the copy variables of agent i for $\boldsymbol{x}_j$ and $\boldsymbol{u}_j$ in the neighborhood $\mathcal{N}_i$.

In ND-DDP, state and control constraints in (3)-(5) are handled using the augmented Lagrangian (AL) method in the DDP layer. Specifically, define the augmented penalty term $C_{i,k} : \mathbb{R}^{\tilde{n}_i} \times \mathbb{R}^{\tilde{m}_i} \times \mathbb{R}^{n_i^{\text{constr}}} \to \mathbb{R}$ as $C_{i,k}(\boldsymbol{x}_{i,k}^a, \boldsymbol{u}_{i,k}^a, \boldsymbol{w}_{i,k}, \zeta_{i,k},) \coloneqq \mathscr{P}(\boldsymbol{w}_{i,k}, \zeta_{i,k}, \tilde{\boldsymbol{b}}_i^a(\boldsymbol{u}_{i,k}^a), \tilde{\boldsymbol{g}}_i^a(\boldsymbol{x}_{i,k}^a), \tilde{\boldsymbol{h}}_i^a(\boldsymbol{x}_{i,k}^a))$, where n_i^{constr} is the total number of constraints for agent i, $\mathscr{P}$ is a penalty function, $\tilde{\boldsymbol{g}}_i^a : \mathbb{R}^{\tilde{n}_i} \to \mathbb{R}^{\tilde{n}_i}$ is an independent state constraint for the augmented state of agent i, $\tilde{\boldsymbol{h}}_i^a : \mathbb{R}^{\tilde{n}_i} \to \mathbb{R}^{\tilde{n}_i}$ is a pair-wise state constraint for the augmented state of agent i, and $\tilde{\boldsymbol{b}}_i^a : \mathbb{R}^{\tilde{m}_i} \to \mathbb{R}^{\tilde{m}_i}$ is a control constraint for the augmented state of agent i, $(\boldsymbol{w}_{i,k}, \zeta_{i,k})$ are the Lagrange multipliers and penalty parameters for the constraints $[\tilde{\boldsymbol{b}}_i^a(\boldsymbol{u}_{i,k}^a), \tilde{\boldsymbol{g}}_i^a(\boldsymbol{x}_{i,k}^a), \tilde{\boldsymbol{h}}_i^a(\boldsymbol{x}_{i,k}^a)] \le 0$. The distributed optimal control problem for ND-DDP is formulated as follows.

Problem 4 (Nested distributed optimal control problem)

$$\min_{\boldsymbol{u}^a} \sum_{i=1}^{M} [J_i(\boldsymbol{x}_i^a, \boldsymbol{u}_i^a) + \sum_{k=0}^{T} C_{i,k}(\boldsymbol{x}_{i,k}^a, \boldsymbol{u}_{i,k}^a)] \tag{25}$$

$$\textit{subject to} \quad \boldsymbol{x}_{i,k+1}^a = \boldsymbol{F}_i^a(\boldsymbol{x}_{i,k}^a, \boldsymbol{u}_{i,k}^a), \qquad \boldsymbol{u}_{i,k}^a = \boldsymbol{y}_{i,k}^a, \quad \forall i \in [M], \forall k = 0, \dots, T-1, \tag{26}$$

$$\boldsymbol{x}_{i,k}^a = \boldsymbol{z}_{i,k}^a, \forall i \in [M], \forall k = 0, \dots, T, \tag{27}$$

where $\boldsymbol{F}_i^a : \mathbb{R}^{\tilde{n}_i} \times \mathbb{R}^{\tilde{m}_i} \to \mathbb{R}^{\tilde{n}_i}$ is a dynamics function for the augmented state of agent i. In particular, the Lagrange multiplier $\boldsymbol{w}_{i,k}$ associated with the constraint $[\tilde{\boldsymbol{b}}_i^a(\boldsymbol{u}_{i,k}^a), \tilde{\boldsymbol{g}}_i^a(\boldsymbol{x}_{i,k}^a), \tilde{\boldsymbol{h}}_i^a(\boldsymbol{x}_{i,k}^a)] \le 0$ is optimized in an inner AL optimization loop.

Although ND-DDP is more scalable than centralized AL-DDP, the computational complexity of ND-DDP still largely depends on the size of the neighborhood set and the number of AL-DDP iterations. The primal, global, and dual update rules of ADMM in ND-DDP can be found in Appendix D.1.

To improve the scalability of the nested architecture of DDP (Saravanos et al., 2023), we unroll the inner AL optimization loop in the original problem formulation described in Appendix C. To form the unrolled AL cost function, we transform the inequality constraints in (3)-(5) using the penalty function defined as $\mathscr{P}_v(t) = \max(0, t+v)^2$, where $v \ge 0$ is a shift parameter. The penalty function $\mathscr{P}_v(t)$ is convex with respect to t. Further remarks on the penalty function $\mathscr{P}_v(t)$ can be found in Appendix C.1. Subsequently, we solve the following problem in a receding horizon fashion.

Problem 5 (Unrolled nested distributed optimal control problem)

$$\min_{\boldsymbol{u}^a} \sum_{i=1}^{M} [J_i(\boldsymbol{x}_i^a, \boldsymbol{u}_i^a)] \tag{28}$$

$$\textit{subject to} \quad \boldsymbol{x}_{i,k+1}^a = \boldsymbol{F}_i^a(\boldsymbol{x}_{i,k}^a, \boldsymbol{u}_{i,k}^a), \quad \mathscr{P}_v(\boldsymbol{b}_i^a(\boldsymbol{u}_{i,k}^a)) = \boldsymbol{0}, \quad \forall i \in [M], \forall k = 0, \dots, T-1, \tag{29}$$

$$\mathscr{P}_v(\boldsymbol{g}_i^a(\boldsymbol{x}_{i,k}^a)) = \boldsymbol{0}, \quad \mathscr{P}_v(\boldsymbol{h}_i^a(\boldsymbol{x}_{i,k}^a)) = \boldsymbol{0}, \quad \forall i \in [M], \forall k = 0, \dots, T, \tag{30}$$

$$\boldsymbol{x}_{i,k}^a = \boldsymbol{z}_{i,k}^a, \forall i \in [M], \forall k = 0, \dots, T, \quad \boldsymbol{u}_{i,k}^a = \boldsymbol{y}_{i,k}^a, \forall i \in [M], \forall k = 0, \dots, T-1. \tag{31}$$

To solve Problem 5 using distributed ADMM, we formulate the AL for Problem 5.

Augmented Lagrangian formulation We define the stage and terminal cost functions with the augmented penalty terms for the nested architecture as

$$\hat{l}_i(\boldsymbol{x}^a_{i,k}, \boldsymbol{u}^a_{i,k}) \coloneqq \sum_{j\in\mathcal{N}_i} \frac{1}{|\mathcal{P}_j|}\Big[l_i(\boldsymbol{x}^i_{j,k}, \boldsymbol{u}^i_{j,k}) + \frac{\rho}{2}\big\|\boldsymbol{x}^i_{j,k} - \boldsymbol{z}^i_{j,k} + \boldsymbol{\lambda}^i_{j,k}/\rho\big\|_2^2 + \frac{\eta}{2}\big\|\boldsymbol{u}^i_{j,k} - \boldsymbol{y}^i_{j,k} + \boldsymbol{\mu}^i_{j,k}/\eta\big\|_2^2$$
$$+ \frac{\alpha}{2}\big\|\mathscr{P}_v(\boldsymbol{b}_j(\boldsymbol{u}^i_{j,k})) + \boldsymbol{\phi}^i_{j,k}/\alpha\big\|_2^2 + \frac{\beta}{2}\big\|\mathscr{P}_v(\boldsymbol{g}_j(\boldsymbol{x}^i_{j,k})) + \mathscr{P}_v(\boldsymbol{h}_{i,j}(\boldsymbol{x}^i_{i,k}, \boldsymbol{x}^i_{j,k})) + \boldsymbol{\psi}^i_{j,k}/\beta\big\|_2^2\Big], \tag{32}$$

$$\hat{\Phi}_i(\boldsymbol{x}^a_{i,T}) \coloneqq \sum_{j\in\mathcal{N}_i} \frac{1}{|\mathcal{P}_j|}\Big[\Phi_j(\boldsymbol{x}^i_{j,T}) + \frac{\rho}{2}\big\|\boldsymbol{x}^i_{j,T} - \boldsymbol{z}^i_{j,T} + \boldsymbol{\lambda}^i_{j,T}/\rho\big\|_2^2$$
$$+ \frac{\beta}{2}\big\|\mathscr{P}_v(\boldsymbol{g}_j(\boldsymbol{x}^i_{j,T})) + \mathscr{P}_v(\boldsymbol{h}_{i,j}(\boldsymbol{x}^i_{i,T}, \boldsymbol{x}^i_{j,T})) + \boldsymbol{\psi}^i_{j,T}/\beta\big\|_2^2\Big], \tag{33}$$

where $(\rho, \eta, \alpha, \beta)$ and $(\boldsymbol{\lambda}^i_{j,k}, \boldsymbol{\mu}^i_{j,k}, \boldsymbol{\phi}^i_{j,k}, \boldsymbol{\psi}^i_{j,k})$ are the penalty parameters and dual variables associated with the constraints in (31), (29), (30). For simplicity, we denote $\boldsymbol{z}^a_{i,k} \coloneqq [(\boldsymbol{z}_{j,k})_{j\in\mathcal{N}_i}]$, $\boldsymbol{y}^a_{i,k} \coloneqq [(\boldsymbol{y}_{j,k})_{j\in\mathcal{N}_i}]$, $\boldsymbol{\lambda}^a_{i,k} \coloneqq [(\boldsymbol{\lambda}^i_{j,k})_{j\in\mathcal{N}_i}]$, $\boldsymbol{\mu}^a_{i,k} \coloneqq [(\boldsymbol{\mu}^i_{j,k})_{j\in\mathcal{N}_i}]$, $\boldsymbol{\phi}^a_{i,k} \coloneqq [(\boldsymbol{\phi}^i_{j,k})_{j\in\mathcal{N}_i}]$, $\boldsymbol{\psi}^a_{i,k} \coloneqq [(\boldsymbol{\psi}^i_{j,k})_{j\in\mathcal{N}_i}]$. Similarly, $\boldsymbol{z}^a_i \coloneqq [(\boldsymbol{z}^a_{i,k})_{k=0}^T]$, $\boldsymbol{y}^a_i \coloneqq [(\boldsymbol{y}^a_{i,k})_{k=0}^{T-1}]$, $\boldsymbol{\lambda}^a_i \coloneqq [(\boldsymbol{\lambda}^a_{i,k})_{k=0}^T]$, $\boldsymbol{\mu}^a_i \coloneqq [(\boldsymbol{\mu}^a_{i,k})_{k=0}^{T-1}]$, $\boldsymbol{\phi}^a_i \coloneqq [(\boldsymbol{\phi}^a_{i,k})_{k=0}^{T-1}]$, $\boldsymbol{\psi}^a_i \coloneqq [(\boldsymbol{\psi}^a_{i,k})_{k=0}^T]$. Finally, $\boldsymbol{z}^a \coloneqq [(\boldsymbol{z}^a_i)_{i\in[M]}]$, $\boldsymbol{y}^a \coloneqq [(\boldsymbol{y}^a_i)_{i\in[M]}]$, $\boldsymbol{\lambda}^a \coloneqq [(\boldsymbol{\lambda}^a_i)_{i\in[M]}]$, $\boldsymbol{\mu}^a \coloneqq [(\boldsymbol{\mu}^a_i)_{i\in[M]}]$, $\boldsymbol{\phi}^a \coloneqq [(\boldsymbol{\phi}^a_i)_{i\in[M]}]$, $\boldsymbol{\psi}^a \coloneqq [(\boldsymbol{\psi}^a_i)_{i\in[M]}]$.

The stage and terminal cost functions in (32) and (33) are divided by $|\mathcal{P}_j|$ to assign equal weights to individual agents as in Problem 1. The AL cost function $\hat{J}_i : \mathbb{R}^{n_i\times|\mathcal{N}_i|\times(T+1-k)} \times \mathbb{R}^{m_i\times|\mathcal{N}_i|\times(T-k)} \to \mathbb{R}$ is defined as $\hat{J}_i(\boldsymbol{x}^a_{i,k:T}, \boldsymbol{u}^a_{i,k:T-1}) \coloneqq \hat{\Phi}_i(\boldsymbol{x}^a_{i,T}) + \sum_{t=k}^{T-1} \hat{l}_i(\boldsymbol{x}^a_{i,t}, \boldsymbol{u}^a_{i,t})$. The corresponding value function is defined as $\hat{V}_i(\boldsymbol{x}^a_{i,k}) \coloneqq \min_{\boldsymbol{u}^a_{i,k:T-1}} \hat{J}_i(\boldsymbol{x}^a_{i,k:T}, \boldsymbol{u}^a_{i,k:T-1})$. By Bellman's principle of optimality, we have $\hat{V}_i(\boldsymbol{x}^a_{i,k}) = \min_{\boldsymbol{u}^a_{i,k:T-1}}[\hat{l}_i(\boldsymbol{x}^a_{i,k}, \boldsymbol{u}^a_{i,k}) + \hat{V}_i(\boldsymbol{x}^a_{i,k+1})]$. The ADMM update rules for the nested architecture can be found in Appendix D.1.

C.1 Penalty function in ND-DDP

Remark 1 (The penalty function) *The penalty function $\mathscr{P}_v$ is analogous to the Powell-Hestenes-Rockafellar (PHR) method for inequality constraints (Rockafellar, 1973). In particular, the PHR AL function involves a penalty term defined as $P_\rho(g(x), \lambda) = \frac{\rho}{2}\max\left(0, g(x) + \frac{\lambda}{\rho}\right)^2$, where $\rho > 0$ is the penalty parameter and λ is the Lagrange multiplier estimate. Here, the term $\frac{\lambda}{\rho}$ in P_ρ corresponds to v of $\mathscr{P}_v(t)$. In practice, we use a fixed constant $v > 0$, where it creates a safety margin and provides a non-zero gradients at the constraint boundary (Haftka & STARNES, 1976), mitigating the stagnation issues of quadratic penalty methods (Nocedal & Wright, 2006).*

D ADMM update rules

D.1 Nested Distributed DDP

Primal updates To perform distributed optimization, Problem 5 can be decomposed into M DDP subproblems with the AL cost function $\hat{J}^{\mathcal{L}}_i$ for each agent i as $\boldsymbol{u}^{a,(n+1)}_i = \arg\min_{\boldsymbol{u}^a_i} \hat{J}_i(\boldsymbol{x}^{a,(n)}_i, \boldsymbol{u}^a_i)$, subject to $\boldsymbol{x}^{a,(n+1)}_{i,k+1} = \boldsymbol{F}_i(\boldsymbol{x}^{a,(n+1)}_{i,k}, \boldsymbol{u}^{a,(n+1)}_{i,k})$ for all time instants k, where the superscript $^{(n)}$ represents the n-th ADMM iteration.

Global and dual updates Subsequently, the global variables in ADMM for each agent i are updated as

$$\boldsymbol{z}^{(n+1)}_{i,k} = \frac{1}{|\mathcal{P}_i|}\sum_{j\in\mathcal{P}_i}[\boldsymbol{x}^{j,(n+1)}_{i,k}] + \frac{1}{\rho}\boldsymbol{\lambda}^{(n)}_{i,k}, \qquad \boldsymbol{y}^{(n+1)}_{i,k} = \frac{1}{|\mathcal{P}_i|}\sum_{j\in\mathcal{P}_i}[\boldsymbol{u}^{j,(n+1)}_{i,T}] + \frac{1}{\eta}\boldsymbol{\mu}^{(n)}_{i,T}, \tag{34}$$

for all time instants k. Finally, we update the dual variables for each agent i as:
(i) $\boldsymbol{\lambda}^{a,(n+1)}_{i,k} = \boldsymbol{\lambda}^{a,(n)}_{i,k} + \hat{\rho}(\boldsymbol{x}^{a,(n+1)}_{i,k} - \boldsymbol{z}^{a,(n+1)}_{i,k})$; (ii) $\boldsymbol{\mu}^{a,(n+1)}_{i,k} = \boldsymbol{\mu}^{a,(n)}_{i,k} + \hat{\eta}(\boldsymbol{u}^{a,(n+1)}_{i,k} - \boldsymbol{y}^{a,(n+1)}_{i,k})$;

(iii) $\boldsymbol{\phi}_{i,k}^{a,(n+1)} = \boldsymbol{\phi}_{i,k}^{a,(n)} + \hat{\alpha}\big(\mathscr{P}_v(\boldsymbol{b}_i^a(\boldsymbol{u}_{i,k}^{a,(n+1)}))\big)$;
(iv) $\boldsymbol{\psi}_{i,k}^{a,(n+1)} = \boldsymbol{\psi}_{i,k}^{a,(n)} + \hat{\beta}\big(\mathscr{P}_v(\boldsymbol{g}_i^a(\boldsymbol{x}_{i,k}^{a,(n+1)})) + \mathscr{P}_v(\boldsymbol{h}_i^a(\boldsymbol{x}_{i,k}^{a,(n+1)}))\big)$, where $\hat{\rho}, \hat{\eta}, \hat{\alpha}, \hat{\beta}$ are scaled penalty parameters depending on the size of the set $|\mathcal{P}_i|$. We perform these primal, dual, and global updates iteratively in ADMM until convergence or the maximum number of iterations is reached.

D.2 Merged Distributed DDP

To perform distributed optimization using ADMM, we formulate the AL for Problem 2.

Augmented Lagrangian formulation In the merged architecture, the stage and terminal cost functions with the augmented penalty terms are defined as

$$\tilde{l}_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}) := l_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}) + \frac{\tilde{\rho}}{2}\big\|\tilde{\boldsymbol{x}}_{i,k}^a - \boldsymbol{z}_{i,k}^a + \boldsymbol{\lambda}_{i,k}/\tilde{\rho}\big\|_2^2 + \frac{\tilde{\eta}}{2}\big\|\boldsymbol{u}_{i,k} - \tilde{\boldsymbol{u}}_{i,k} + \boldsymbol{\mu}_{i,k}/\tilde{\eta}\big\|_2^2 + \frac{\tilde{\sigma}}{2}\big\|\boldsymbol{x}_{i,k} - \tilde{\boldsymbol{x}}_{i,k} + \boldsymbol{\xi}_{i,k}/\tilde{\sigma}\big\|_2^2, \tag{35}$$

$$\tilde{\Phi}_i(\boldsymbol{x}_{i,T}) := \Phi_i(\boldsymbol{x}_{i,T}) + \frac{\tilde{\rho}}{2}\big\|\tilde{\boldsymbol{x}}_{i,T}^a - \boldsymbol{z}_{i,T}^a + \boldsymbol{\lambda}_{i,T}/\tilde{\rho}\big\|_2^2 + \frac{\tilde{\sigma}}{2}\big\|\boldsymbol{x}_{i,T} - \tilde{\boldsymbol{x}}_{i,T} + \boldsymbol{\xi}_{i,T}/\tilde{\sigma}\big\|_2^2, \tag{36}$$

where $(\tilde{\rho}, \tilde{\eta}, \tilde{\sigma})$ and $(\boldsymbol{\lambda}_{i,k}, \boldsymbol{\mu}_{i,k}, \boldsymbol{\xi}_{i,k})$ are the penalty parameters and dual variables associated with the constraints in (11)-(12). For simplicity, we denote $\boldsymbol{\lambda}_i := [(\boldsymbol{\lambda}_{i,k})_{k=0}^T]$, $\boldsymbol{\mu}_i := [(\boldsymbol{\mu}_{i,k})_{k=0}^{T-1}]$, $\boldsymbol{\xi}_i := [(\boldsymbol{\xi}_{i,k})_{k=0}^T]$, $\tilde{\boldsymbol{x}}_i^a := [(\tilde{\boldsymbol{x}}_{i,k}^a)_{k=0}^T]$, $\tilde{\boldsymbol{u}}_i := [(\tilde{\boldsymbol{u}}_{i,k})_{k=0}^{T-1}]$, $\tilde{\boldsymbol{x}}_i := [(\tilde{\boldsymbol{x}}_{i,k})_{k=0}^T]$. Similarly, $\boldsymbol{\lambda} := [(\boldsymbol{\lambda}_i)_{i\in[M]}]$, $\boldsymbol{\mu} := [(\boldsymbol{\mu}_i)_{i\in[M]}]$, $\boldsymbol{\xi} := [(\boldsymbol{\xi}_i)_{i\in[M]}]$, $\tilde{\boldsymbol{x}}^a := [(\tilde{\boldsymbol{x}}_i^a)_{i\in[M]}]$, $\tilde{\boldsymbol{u}} := [(\tilde{\boldsymbol{u}}_i)_{i\in[M]}]$, $\tilde{\boldsymbol{x}} := [(\tilde{\boldsymbol{x}}_i)_{i\in[M]}]$. The AL cost function $\tilde{J}_i : \mathbb{R}^{n_i\times(T+1-k)} \times \mathbb{R}^{m_i\times(T-k)} \to \mathbb{R}$ is defined as $\tilde{J}_i(\boldsymbol{x}_{i,k:T}, \boldsymbol{u}_{i,k:T-1}) := \tilde{\Phi}_i(\boldsymbol{x}_{i,T}) + \sum_{t=k}^{T-1} \tilde{l}_i(\boldsymbol{x}_{i,t}, \boldsymbol{u}_{i,t})$. We define the corresponding value function as $\tilde{V}_i(\boldsymbol{x}_{i,k}) := \min_{\boldsymbol{u}_{i,k:T-1}} \tilde{J}_i(\boldsymbol{x}_{i,k:T}, \boldsymbol{u}_{i,k:T-1})$. By Bellman's principle of optimality, we have $\tilde{V}_i(\boldsymbol{x}_{i,k}) = \min_{\boldsymbol{u}_{i,k:T-1}}[\tilde{l}_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k}) + \tilde{V}_i(\boldsymbol{x}_{i,k+1})]$.

Primal updates We decompose Problem 2 into M subproblems with AL cost function $\tilde{J}_i$ for each agent i as $\boldsymbol{u}_i^{(n+1)} = \arg\min_{\boldsymbol{u}_i} \tilde{J}_i(\boldsymbol{x}_i^{(n)}, \boldsymbol{u}_i)$ subject to $\boldsymbol{x}_{i,k+1}^{(n+1)} = \boldsymbol{F}_i(\boldsymbol{x}_{i,k}^{(n+1)}, \boldsymbol{u}_{i,k}^{(n+1)})$, for all time instants k, which is solved using DDP.

Safe updates The safe variables $\tilde{x}_i^a, \tilde{u}_i$ are updated by imposing state avoidance and control limit constraints in (8)-(10) given the DDP state and control trajectory for each agent i. In particular, the safe variables are updated as

$$(\tilde{\boldsymbol{x}}_i^{(n+1)}, \tilde{\boldsymbol{x}}_i^{a,(n+1)}, \tilde{\boldsymbol{u}}_i^{(n+1)}) = \arg\min_{\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_i^a, \tilde{\boldsymbol{u}}_i} \tilde{J}_i(\boldsymbol{x}_i^{(n+1)}, \boldsymbol{u}_i^{(n+1)}) \tag{37}$$

$$\text{subject to } \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \le 0, \boldsymbol{h}_i^a(\tilde{\boldsymbol{x}}_{i,k}^a)) \le 0, \boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \le 0, \tag{38}$$

for all $i \in [M]$, and k. The problem (37) can be split into safe state and control update, and decomposed with respect to time steps as

$$(\tilde{\boldsymbol{x}}_{i,k}^{(n+1)}, \tilde{\boldsymbol{x}}_{i,k}^{a,(n+1)}) = \arg\min_{\tilde{\boldsymbol{x}}_{i,k}, \tilde{\boldsymbol{x}}_{i,k}^a} \frac{\tilde{\rho}}{2}\big\|\tilde{\boldsymbol{x}}_{i,k}^a - \boldsymbol{z}_{i,k}^{a,(n)} + \boldsymbol{\lambda}_{i,k}^{(n)}/\tilde{\rho}\big\|_2^2 + \frac{\tilde{\sigma}}{2}\big\|\boldsymbol{x}_{i,k}^{(n+1)} - \tilde{\boldsymbol{x}}_{i,k} + \boldsymbol{\xi}_{i,k}^{(n)}/\tilde{\sigma}\big\|_2^2$$
$$\text{subject to } \boldsymbol{g}_i(\tilde{\boldsymbol{x}}_{i,k}) \le 0, \boldsymbol{h}_i^a(\tilde{\boldsymbol{x}}_{i,k}^a) \le 0, \forall i \in [N], \forall k, \tag{39}$$

$$\tilde{\boldsymbol{u}}_{i,k}^{(n+1)} = \arg\min_{\tilde{\boldsymbol{u}}_{i,k}} \frac{\tilde{\eta}}{2}\big\|\boldsymbol{u}_{i,k}^{(n+1)} - \tilde{\boldsymbol{u}}_{i,k} + \boldsymbol{\mu}_{i,k}^{(n)}/\tilde{\eta}\big\|_2^2, \quad \text{subject to } \boldsymbol{b}_i(\tilde{\boldsymbol{u}}_{i,k}) \le 0, \quad \forall i \in [N], \forall k. \tag{40}$$

Global and dual updates Subsequently, we update the global variables as follows.

$$\boldsymbol{z}_{i,k}^{(n+1)} = \frac{1}{|\mathcal{P}_i|}\sum_{j\in\mathcal{P}_i}\big[\tilde{\boldsymbol{x}}_i^{j,(n+1)}\big] + \frac{1}{\tilde{\rho}}\boldsymbol{\lambda}_{i,k}^{(n)}, \quad \boldsymbol{z}_{i,k}^{a,(n+1)} = \big[(\boldsymbol{z}_{i,k}^{(n+1)})_{i\in\mathcal{N}_i}\big], \quad \forall i \in [N], \forall k \tag{41}$$

The dual variables are updated as

$$\boldsymbol{\lambda}_{i,k}^{(n+1)} = \boldsymbol{\lambda}_{i,k}^{(n)} + \tilde{\rho}\big(\tilde{\boldsymbol{x}}_{i,k}^{a,(n+1)} - \boldsymbol{z}_{i,k}^{a,(n+1)}\big), \quad \boldsymbol{\mu}_{i,k}^{(n+1)} = \boldsymbol{\mu}_{i,k}^{(n)} + \tilde{\eta}\big(\tilde{\boldsymbol{u}}_{i,k}^{(n+1)} - \boldsymbol{u}_{i,k}^{(n+1)}\big), \tag{42}$$

$$\boldsymbol{\xi}_{i,k}^{(n+1)} = \boldsymbol{\xi}_{i,k}^{(n)} + \tilde{\sigma}\big(\tilde{\boldsymbol{x}}_{i,k}^{(n+1)} - \boldsymbol{x}_{i,k}^{(n+1)}\big), \quad \forall i \in [M], \forall k. \tag{43}$$

E Algorithm

We denote the problem instance by a tuple I defined as $I^{\text{prob}} := (T, dt, (\boldsymbol{F}_i, n_i, m_i, \boldsymbol{x}_{i,0}, \boldsymbol{x}_{i,T}, \boldsymbol{b}_i, \boldsymbol{g}_i, \{\boldsymbol{h}_{i,j}\}_{j\in\mathcal{N}_i})_{i=1}^{M})$, where dt is a discrete time interval. The algorithm configuration I^{alg} is defined as $I^{\text{alg}} := ((J_i, \rho, \eta, \alpha, \beta, \tilde{\rho}, \tilde{\sigma}, \tilde{\eta})_{i=1}^{M}, H, f^{\text{replan}}, N^{\text{MPC}}, N^{\text{ADMM}}, N^{\text{DDP}}, \epsilon^{\text{ADMM}}, \epsilon^{\text{DDP}})$, where H is the prediction horizon, f^{replan} is the replanning frequency, N^{plan} is the number of planning steps, N^{ADMM} is the maximum number of ADMM iterations, N^{DDP} is the maximum number of DDP iterations, ϵ^{ADMM} is the ADMM convergence threshold, and ϵ^{DDP} is the DDP convergence threshold. Subsequently, we introduce the MPD-DDP algorithm.

Algorithm 1 Model Predictive Distributed DDP

Input: Problem instance I^{prob}, algorithm configuration I^{alg}, and replanning interval $\Delta^{\text{replan}} \leftarrow \lfloor 1/(f^{\text{replan}} \cdot dt) \rfloor$

$\boldsymbol{x}_i', \boldsymbol{u}_i' \leftarrow$ Solve Problem 1 without constraints $\forall i \in [M]$
Compute $\mathcal{N}_i^{(0)}, \mathcal{P}_i^{(0)}, \forall i \in [M]$
Initialize variables: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i'$, $\boldsymbol{u}_i \leftarrow \boldsymbol{u}_i'$, $\tilde{\boldsymbol{x}}_i^a \leftarrow [(\boldsymbol{x}_j')_{j\in\mathcal{N}_i^{(0)}}]$, $\tilde{\boldsymbol{u}}_i \leftarrow \boldsymbol{u}_i'$, $\boldsymbol{z}_i^a \leftarrow \tilde{\boldsymbol{x}}_i^a$, $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{0}$, $\boldsymbol{\mu}_i \leftarrow \boldsymbol{0}$, $\boldsymbol{\xi}_i \leftarrow \boldsymbol{0}$, $\forall i \in [M]$
for $o = 1, \ldots, N^{\text{plan}}$ **do** ▷ Closed-loop iterations
  Measure current state $\boldsymbol{x}_{i,0}$, and compute $\mathcal{N}_i^{(o)}, \mathcal{P}_i^{(o)}, \forall i \in [M]$
  if $\mathcal{N}_i^{(o)} \neq \mathcal{N}_i^{(o-1)}$ for any $i \in [M]$ **then**
    Reset $\tilde{\boldsymbol{x}}^a, \tilde{\boldsymbol{u}}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\xi} \leftarrow \boldsymbol{0}$
    Reinitialize $\boldsymbol{x}_i^a \leftarrow [(\boldsymbol{x}_j)_{j\in\mathcal{N}_i^{(0)}}]$, $\boldsymbol{z}_i^a \leftarrow \boldsymbol{x}_i^a, \forall i \in [M]$
  end if
  Reinitialize $\boldsymbol{z}^a \leftarrow \boldsymbol{x}^a$
  Initialize DDP using $\boldsymbol{x}, \boldsymbol{u}$, and initialize ADMM using $\boldsymbol{z}^a, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\xi}$
  for $l^{\text{ADMM}} = 1, \ldots, N^{\text{ADMM}}$ **do** ▷ ADMM iterations
    $\boldsymbol{u}_i \leftarrow \arg\min_{\boldsymbol{u}_i} \tilde{J}_i(\boldsymbol{x}_i^{(n)}, \boldsymbol{u}_i)$ subject to dynamics in parallel $\forall i \in [M]$ ▷ DDP
    $\boldsymbol{x}_i \leftarrow$ Propagate dynamics $\boldsymbol{F}_i$ using $\boldsymbol{u}_i, \forall i \in [M]$
    $(\tilde{\boldsymbol{x}}_i^a, \tilde{\boldsymbol{u}}_i) \leftarrow$ Solve (39)-(40) in parallel $\forall i \in [M], \forall k$ ▷ Safe update
    Communicate $\tilde{\boldsymbol{x}}_i^j$ from all $j \in \mathcal{P}_i^{(o)} \setminus \{i\}$ to each $i \in [M]$
    $\boldsymbol{z}_i \leftarrow$ Global update, $\forall i \in [M]$
    Communicate $\boldsymbol{z}_j$ from all $j \in \mathcal{N}_i^{(o)} \setminus \{i\}$ to each $i \in [M]$
    $\boldsymbol{z}_i^a \leftarrow [(\boldsymbol{z}_i)_{i\in\mathcal{N}_i^{(o)}}], \forall i \in [M]$
    $\boldsymbol{\lambda}_i, \boldsymbol{\mu}_i, \boldsymbol{\xi}_i \leftarrow$ Dual update, $\forall i \in [M]$
  end for
  Apply $\boldsymbol{u}_{i,0:\Delta^{\text{replan}}}$ to the system, $\forall i \in [M]$
  Shift $\boldsymbol{x}, \boldsymbol{u}, \tilde{\boldsymbol{x}}^a, \tilde{\boldsymbol{u}}, \boldsymbol{z}^a, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\xi}$ by Δ^{replan} steps
end for

F Input features of the permutation invariant Transformer policy

We construct a set-valued input feature composed of $1 + |\mathcal{N}_i| + N^{\text{obs}}$ tokens for each agent i, where N^{obs} is the number of obstacles. In particular, we consider three blocks for the input feature: Self, obstacle, and neighbor blocks. Let N^{cc} be the number of linear control constraints. Let N^{pos} be the dimensionality of the position. Let $p_{i,0} \in \mathbb{R}^{N^{\text{pos}}}$ be the current position of agent i. Let $\text{vec}(p_{i,0}) \in \mathbb{R}^{N^{\text{pos}}\cdot(T+1)}$ be the current position of agent i repeated over the prediction horizon $T+1$.

The self block is a single vector concatenating the current state $\boldsymbol{x}_{i,0}$, target state $\boldsymbol{x}_i^{\text{ref}}$, the difference between the target and the current state $\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_{i,0}$, the control limits $\boldsymbol{u}_i^{\min}, \boldsymbol{u}_i^{\max} \in \mathbb{R}^{m_i}$, the vectorized linear control constraints $\text{vec}(A_i^{cc}) \in \mathbb{R}^{N^{cc} \cdot m_i}, b_i^{cc} \in \mathbb{R}^{N^{cc}}$, and the one-hot vector representation of the self block $[1, 0, 0]$.

The obstacle block is composed of N^{obs} vectors, where each vector is represented as a concatenation of the obstacle position for all time instants $\text{vec}(p_c^{\text{obs}}) \in \mathbb{R}^{N^{\text{pos}} \cdot (T+1)}$, the safety distance between the obstacle and the agent $d^{\text{obs}} \in \mathbb{R}$, the relative difference between the obstacle and the self agent $\text{vec}(p_c^{\text{obs}}) - \text{vec}(p_{i,0})$, and the one-hot vector representation of the obstacle block $[0, 1, 0]$.

The neighbor block consists of $|\mathcal{N}_i|$ vectors. Let $j \in \mathcal{N}_i$, then each vector is represented as a concatenation of the neighbor j's current state $\boldsymbol{x}_{j,0}$, target state $\boldsymbol{x}_j^{\text{ref}}$, the difference between the target and the current state $\boldsymbol{x}_j^{\text{ref}} - \boldsymbol{x}_{j,0}$, the safety distance between the neighbor and the self agent d_{safe}, the relative difference between the state vectors of the neighbor and the self agent $\boldsymbol{x}_j - \boldsymbol{x}_i$, and the one-hot representation of the neighbor block $[0, 0, 1]$.

G Permutation invariant Transformer model architecture

G.1 Attention

Suppose we have an input token embedding $X^t \in \mathbb{R}^{N^t \times d^{\text{emb}}}$, where N^t is the number of input tokens and d^{emb} is the embedding dimension. The learnable weight matrices for query, key, and value are denoted by $W^Q \in \mathbb{R}^{d^{\text{emb}} \times d^q}$, $W^K \in \mathbb{R}^{d^{\text{emb}} \times d^q}$, and $W^V \in \mathbb{R}^{d^{\text{emb}} \times d^v}$, where d^q is a key/query dimension and d^v is a value dimension. We denote a query, key, and value matrices by $\mathcal{Q} = X^t W^Q \in \mathbb{R}^{N^t \times d^q}$, $\mathcal{K} = X^t W^K \in \mathbb{R}^{N^t \times d^q}$, and $\mathcal{V} = X^t W^V \in \mathbb{R}^{N^t \times d^v}$. We define an attention function as $\text{Att}(\mathcal{Q}, \mathcal{K}, \mathcal{V}; \omega) \coloneqq \omega(\mathcal{Q}\mathcal{K}^\top / \sqrt{d^q})\mathcal{V} \in \mathbb{R}^{N^t \times d^v}$, where ω is an activation function. The similarity matrix $\mathcal{Q}\mathcal{K}^\top$ computes the similarity between every pair of input tokens, and the final weighted sum of the value matrix aims to score the most contextually relevant token.

G.2 Multi-head attention

Multi-head attention (MHA) introduced in Vaswani et al. (2017) is an extension of the attention function that performs parallel attention computation using N^{head} heads. The input token matrix X^t is projected N^{head} times using different learnable weight matrices $W_i^Q, W_i^K \in \mathbb{R}^{d^{\text{emb}} \times d_i^q}$, $W_i^V \in \mathbb{R}^{d^{\text{emb}} \times d_i^v}$ for each head: $\mathcal{Q}_i = X^t W_i^Q$, $\mathcal{K}_i = X^t W_i^K$, $\mathcal{V}_i = X^t W_i^V$. Each head is computed as $h_i = \text{Att}(\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i; \omega_i) \in \mathbb{R}^{N^t \times d_i^v}$, and the multi-head attention function is defined as $\text{MHA}((\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i)_{i=1}^{N^{\text{head}}}; W^O, (\omega_i)_{i=1}^{N^{\text{head}}}) = [h_1, \ldots, h_{N^{\text{head}}}]W^O$, where $W^O \in \mathbb{R}^{N^{\text{head}} \cdot d_i^v \times d^v}$. In the original architecture, dimension hyperparameters are chosen as $d_i^v = d^v / N^{\text{head}}$, $d_i^q = d^q / N^{\text{head}}$. We use the softmax function as the activation function ω_i for all $i = 1, \ldots, N^{\text{head}}$.

G.3 Architecture overview

In Fig. 4, we visualize the overall architecture of the permutation invariant transformer that we employ in our experiments. We use three layers of permutation equivariant encoders and two layers of permutation invariant decoders. Each encoding layer is a composition of multi-head self attention, row-wise feed forward layers, layer normalization layers, and skip connections. The decoder layer is analogous to the encoder layer, except for the multi-head cross attention layer in place of the self attention layer. For the multi-head cross attention layer, key and value are constructed using the output of the encoder, and we use the learned queries. MLP layers denoted as $\text{MLP}_{\text{block1}}$ and $\text{MLP}_{\text{block}N^{\text{block}}}$ are feature embedding layers. The final output of the permutation invariant transformer model is the control sequence over the horizon T.

G.4 Feature embedder

We begin by embedding the input problem context feature vectors $\hat{x}_i \in \mathbb{R}^{d^{\text{feat}}}$, which is a concatenation of (self, obstacle, neighbor) feature vectors: $\hat{x}_i = [\hat{x}_i^{\text{self}}, \hat{x}_i^{\text{obs}}, \hat{x}_i^{\text{neighbor}}]$, for each agent $i \in [M]$. Let $X = [\hat{x}_1, \ldots, \hat{x}_M] \in \mathbb{R}^{M \times d^{\text{feat}}}$ be an input feature matrix. We denote $X_{\text{self}} = [\hat{x}_1^{\text{self}}, \ldots, \hat{x}_M^{\text{self}}]$,

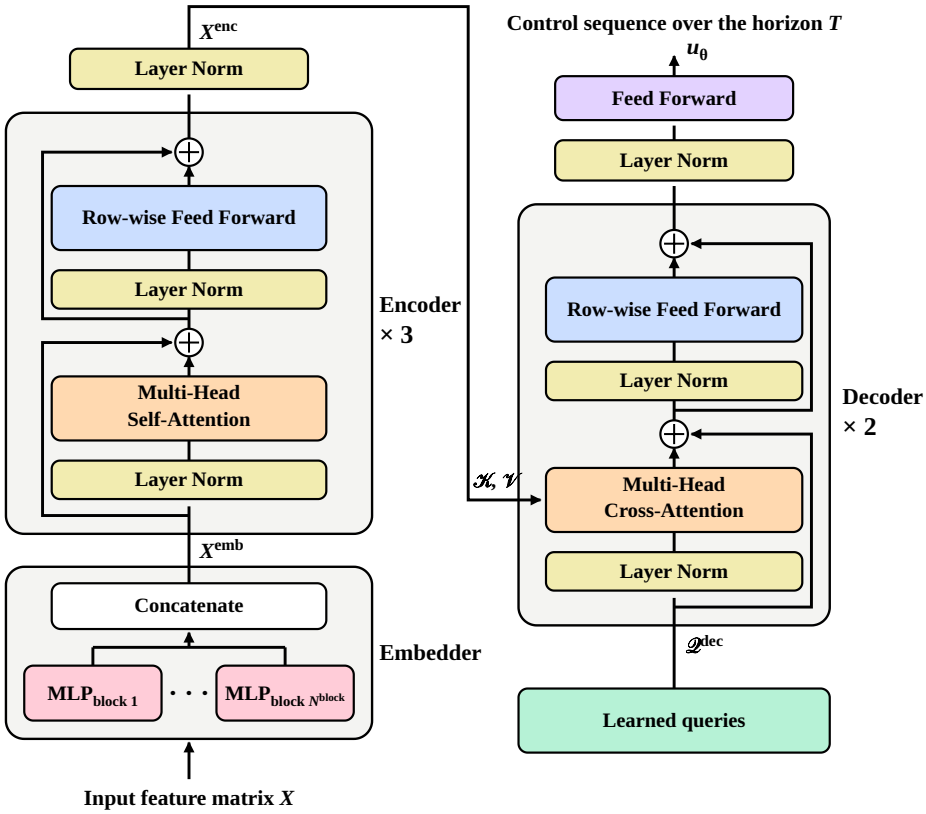


Figure 4: Permutation invariant transformer architecture overview.

$X_{\text{obs}} = [\hat{x}_1^{\text{obs}}, \ldots, \hat{x}_M^{\text{obs}}]$, and $X_{\text{neighbor}} = [\hat{x}_1^{\text{neighbor}}, \ldots, \hat{x}_M^{\text{neighbor}}]$. For each agent i, the embedder generates $N^t = 1 + N^{\text{obs}} + |\mathcal{N}_i|$ tokens, where a single token belongs to the self-agent block, N^{obs} tokens belong to the obstacle block, and $|\mathcal{N}_i|$ tokens belong to the neighbor block:

$$\begin{aligned}\text{Embedder}(X) &= [\text{MLP}_{\text{self}}(X_{\text{self}}), \text{MLP}_{\text{obs}}(X_{\text{obs}}), \text{MLP}_{\text{neighbor}}(X_{\text{neighbor}})] \\ &= [X_{\text{self}}^{\text{emb}}, X_{\text{obs}}^{\text{emb}}, X_{\text{neighbor}}^{\text{emb}}] = X^{\text{emb}} \in \mathbb{R}^{M \times N^t \times d^{\text{emb}}},\end{aligned} \tag{44}$$

where MLP is a multi-layer perceptron. Here, we refer $[X_{\text{self}}^{\text{emb}}, X_{\text{obs}}^{\text{emb}}, X_{\text{neighbor}}^{\text{emb}}]$ to (self, obstacle, neighbor) blocks. In our experiments, we use two layers for each MLP_{self}, MLP_{obs}, $\text{MLP}_{\text{neighbor}}$ with $d^{\text{emb}} = 128$ output dimension and GELU activation functions (Hendrycks & Gimpel, 2016).

G.5 Encoder

Given the embedding X^{emb}, the encoder layer computes the following:

$$\text{Enc}(X^{\text{emb}}) = \tilde{X}^{\text{emb}} + \text{rFF}(\text{LN}(\tilde{X}^{\text{emb}})), \tag{45}$$

where

$$\tilde{X}^{\text{emb}} = X^{\text{emb}} + \text{MHA}^{\text{SA}}((\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i)_{i=1}^{N^{\text{head}}}; W^O, (\omega_i)_{i=1}^{N^{\text{head}}}) \tag{46}$$

Here, LN is a layer normalization layer (Ba et al., 2016), rFF is a row-wise feedforward layer. Key, query, and value matrices are computed as $\mathcal{Q}_i = \text{LN}(X^{\text{emb}})W_i^Q$, $\mathcal{K}_i = \text{LN}(X^{\text{emb}})W_i^K$, $\mathcal{V}_i = \text{LN}(X^{\text{emb}})W_i^V$, where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d^{\text{emb}} \times d^{\text{emb}}}$ are learnable weight matrices for all $i = 1, \ldots, N^{\text{head}}$. Notice that MHA^{SA} in (46) is a multi-head self attention layer. The activation function ω_i is a softmax function for all $i = 1, \ldots, N^{\text{head}}$. In our experiments, we use three encoder layers with $d^{\text{emb}} = 128$ and $N^{\text{head}} = 4$:

$$\text{Encoder}(X^{\text{emb}}) = \text{Enc}(\text{Enc}(\text{Enc}(X^{\text{emb}}))) = X^{\text{enc}} \in \mathbb{R}^{M \times N^t \times d^{\text{emb}}}, \tag{47}$$

We use two MLP layers with 256 intermediate dimensions, 128 output dimensions for the row-wise feedforward layer for each encoder layer, and GELU activation functions. See Appendix G.1 for the definition of the multi-head attention function MHA.

G.6 Decoder

We design the decoder to return a sequence with a fixed length T, the prediction horizon. Let $\mathcal{Q}^{\text{dec}} \in \mathbb{R}^{M \times T \times d^{\text{emb}}}$ be the learned queries, which does not depend on the input X. Given the encoder output X^{enc}, the decoder layer computes the following:

$$\text{Dec}(X^{\text{enc}}) = \tilde{\mathcal{Q}} + \text{rFF}(\text{LN}_f(\tilde{\mathcal{Q}})), \tag{48}$$

where

$$\tilde{\mathcal{Q}} = \mathcal{Q}^{\text{dec}} + \text{MHA}^{\text{cross}}((\mathcal{Q}_i^{\text{dec}}, \mathcal{K}_i^{\text{dec}}, \mathcal{V}_i^{\text{dec}})_{i=1}^{N^{\text{head}}}; W_{\text{dec}}^{O}, (\omega_i)_{i=1}^{N^{\text{head}}}). \tag{49}$$

Here, $\mathcal{Q}_i^{\text{dec}} = \text{LN}_q(\mathcal{Q}^{\text{dec}})W_i^{\text{dec},Q}$, $\mathcal{K}_i^{\text{dec}} = \text{LN}_{kv}(X^{\text{enc}})W_i^{\text{dec},K}$, $\mathcal{V}_i^{\text{dec}} = \text{LN}_{kv}(X^{\text{enc}})W_i^{\text{dec},V}$, where $W_i^{\text{dec},Q}, W_i^{\text{dec},K}, W_i^{\text{dec},V} \in \mathbb{R}^{d^{\text{emb}} \times d^{\text{emb}}}$ are learnable weight matrices for all $i = 1, \ldots, N^{\text{head}}$. Notice that $\text{MHA}^{\text{cross}}$ in (49) is a multi-head cross attention layer. The activation function ω_i is a softmax function for all $i = 1, \ldots, N^{\text{head}}$. In our experiments, we use two decoder layers with $d^{\text{emb}} = 128$ and $N^{\text{head}} = 4$:

$$\text{Decoder}(X^{\text{enc}}) = \text{FF}(\text{LN}_d(\text{Dec}(\text{Dec}(X^{\text{enc}})))) = X^{\text{out}} \in \mathbb{R}^{M \times T \times m}. \tag{50}$$

We use two MLP layers with 128 dimensions for the row-wise feedforward layer for each decoder layer and GELU activation functions. The outermost feedforward network in (50) maps $\mathbb{R}^{d^{\text{emb}}} \to \mathbb{R}^m$ with a single layer, where m is the control dimension.

H Proofs

Formally, let $A = (a_1, \ldots, a_K)$ be an input tuple where each $a_i \in \mathcal{A}$. Let S_K be the symmetric group of all possible permutations of K indices.

Definition 1 *A function f is permutation equivariant if $f(\Pi \cdot A) = \Pi \cdot f(A)$ for any permutation $\Pi \in S_K$.*

Definition 2 *A function f is permutation invariant if $f(\Pi \cdot A) = f(A)$ for any permutation $\Pi \in S_K$.*

We refer to the vectors in the token axis in the embedded vector $X^{\text{emb}} \in \mathbb{R}^{M \times N^t \times d^{\text{emb}}}$ as tokens, *i.e.*, $X^{\text{emb}} = [X_1^{\text{tok}}, \ldots, X_{N^t}^{\text{tok}}]$, where $X_i^{\text{tok}} \in \mathbb{R}^{M \times d^{\text{emb}}}$ for all $i = 1, \ldots, N^t$.

Lemma 1 *The final encoder (47) is a permutation equivariant mapping with respect to the ordering of tokens $(X_i^{tok})_{i=1}^{N^t}$.*

Proof *We show that the encoder layer in (45) is permutation equivariant. It is sufficient to show that the self attention function in (46) is permutation equivariant. Let $\Pi \in \{0,1\}^{N^t \times N^t}$ be an arbitrary permutation matrix on the token axis. By the orthogonality of Π, we have $\Pi^\top \Pi = I$, where $I \in \mathbb{R}^{N^t \times N^t}$ is an identity matrix. For notational brevity, we denote $\text{SA}(X^{emb}) := \text{MHA}^{SA}((\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i)_{i=1}^{N^{head}}; W^O, (\omega_i)_{i=1}^{N^{head}})$ with $N^{head} = 1$. The self attention in (46) with X^{emb} yields*

$$\text{SA}(X^{emb}) = \omega(X^{emb} W^Q {W^K}^\top {X^{emb}}^\top / \sqrt{d^{emb}}) X^{emb} W^V W^O \tag{51}$$

The self attention with ΠX^{emb} yields

$$\begin{aligned}
\text{SA}(\Pi X^{emb}) &= \omega(\Pi X^{emb} W^Q {W^K}^\top {X^{emb}}^\top \Pi^\top / \sqrt{d^{emb}}) \Pi X^{emb} W^V W^O \\
&= \Pi \omega(X^{emb} W^Q {W^K}^\top {X^{emb}}^\top / \sqrt{d^{emb}}) \underbrace{\Pi^\top \Pi}_{=I} X^{emb} W^V W^O \\
&= \Pi \text{SA}(X^{emb}),
\end{aligned} \tag{52}$$

which shows the permutation equivariance of the multi-head self attention layer in (46) by Definition 1. Here, we omit the layer normalization in the self attention layer as it does not alter the ordering of dimensions of X^{emb}. Furthermore, the row-wise feedforward layer, matrix addition, and composition of permutation equivariant functions in (45)-(46) preserve permutation equivariance. It concludes that the final encoder (47) is a permutation equivariant mapping.

Lemma 2 *The final decoder (50) is a permutation invariant mapping with respect to the ordering of tokens* $(X_i^{tok})_{i=1}^{N^t}$.

Proof *We show that the decoder layer in (48) is permutation invariant. It is sufficient to show that the cross attention layer in (49) is permutation invariant. For notational simplicity, we denote* $\mathrm{CA}(X^{enc}, \mathcal{Q}^{dec}) := \mathrm{MHA}^{cross}((\mathcal{Q}_i, \mathcal{K}_i, \mathcal{V}_i)_{i=1}^{N^{head}}; W^O, (\omega_i)_{i=1}^{N^{head}})$ *with* $N^{head} = 1$. *The cross attention in (49) with* X^{enc} *and* $\mathcal{Q}^{dec}$ *yields*

$$\mathrm{CA}(X^{enc}, \mathcal{Q}^{dec}) = \omega(\mathcal{Q}^{dec} W^Q {W^K}^\top {X^{enc}}^\top / \sqrt{d^{emb}}) X^{enc} W^V W^O \tag{53}$$

The cross attention with the permutated input ΠX^{enc} *yields*

$$\begin{aligned}\mathrm{CA}(\Pi X^{enc}, \mathcal{Q}^{dec}) &= \omega(\mathcal{Q}^{dec} W^Q {W^K}^\top {X^{enc}}^\top \Pi^\top / \sqrt{d^{emb}}) \Pi X^{enc} W^V W^O \\ &= \omega(\mathcal{Q}^{dec} W^Q {W^K}^\top {X^{enc}}^\top / \sqrt{d^{emb}}) \underbrace{\Pi^\top \Pi}_{=I} X^{enc} W^V W^O \\ &= \mathrm{CA}(X^{enc}, \mathcal{Q}^{dec}),\end{aligned} \tag{54}$$

which shows the permutation invariance of the multi-head cross attention layer in (49) by Definition 2. Here, we omit the layer normalization in the cross attention layer as it does not alter the ordering of dimensions of X^{enc}. *Furthermore, the row-wise feedforward layer, matrix addition, standard feedforward layer, and composition of permutation invariance functions in (48)-(50) preserve permutation invariance. It concludes that the final encoder (50) is a permutation invariance mapping.*

Subsequently, we show that the final model is permutation invariant with respect to the ordering of input tokens. We denote the final permutation invariant transformer model by

$$\hat{y}_\theta(X) := \mathrm{Decoder}(\mathrm{Encoder}(\mathrm{Embedder}(X))). \tag{55}$$

Lemma 3 *The final permutation invariant transformer model (55) is a permutation invariant mapping with respect to the ordering of tokens* $(X_i^{tok})_{i=1}^{N^t}$.

Proof *Given* $\mathrm{Embedder}(X) = X^{emb} = [X_1^{tok}, \ldots, X_{N^t}^{tok}]$, *consider an embedded matrix* X^{emb} *multiplied by an arbitrary permutation matrix* $\Pi \in \{0,1\}^{N^t \times N^t}$ *on the token axis:*

$$\begin{aligned}\mathrm{Decoder}(\mathrm{Encoder}(\Pi \cdot \mathrm{Embedder}(X))) &= \mathrm{Decoder}(\Pi \cdot \mathrm{Encoder}(\mathrm{Embedder}(X))) \\ &\qquad \text{(By Lemma 1)} \\ &= \mathrm{Decoder}(\mathrm{Encoder}(\mathrm{Embedder}(X))) \quad \text{(By Lemma 2)} \\ &= \hat{y}_\theta(X),\end{aligned} \tag{56}$$

which concludes the proof.

Let $X = [\hat{x}_1, \ldots, \hat{x}_M] \in \mathbb{R}^{M \times d^{\text{feat}}}$ be an input feature matrix, where M is the number of agents, and $\hat{x}_i \in \mathbb{R}^{d^{\text{feat}}}$ is a problem context vector for each agent $i \in [M]$. The model parameters θ are shared by all agents, which leads to the following permutation equivariant property with respect to the ordering of the agents.

Remark 2 *Since* θ *is shared by all agents* $i \in [M]$, *suppose that* $\hat{y}_\theta(X) = [\hat{y}_\theta(\hat{x}_1), \ldots, \hat{y}_\theta(\hat{x}_M)]$. *Let* $\Pi^a \in \{0,1\}^{M \times M}$ *be an arbitrary permutation matrix associated with the agent index permutation* π^a, *where* $\Pi^a_{i,j} = 1$, *if* $i = \pi^a(j)$, *otherwise* 0. *It follows that* $\Pi^a X = [\hat{x}_{\pi^a(1)}, \ldots, \hat{x}_{\pi^a(M)}]$, *which implies that* $\hat{y}_\theta(\Pi^a X) = [\hat{y}_\theta(\hat{x}_{\pi^a(1)}), \ldots, \hat{y}_\theta(\hat{x}_{\pi^a(M)})] = \Pi^a \hat{y}_\theta(X)$.

Proposition 1 *If there exist continuous optimal control policies* $\pi_i^{policy} : \mathbb{R}^{d^{feat}} \to \mathbb{R}^{m_i \times T}$ *for all* $i \in [M]$ *in the distributed optimal control problem in Problem 2, the permutation invariant transformer model (55) is a universal approximator of* π_i^{policy} *for all* $i \in [M]$.

Proof *By description, any control policy for Problem 2 is required to: i) generalize over an arbitrary number of agents* M, *neighbors* $|\mathcal{N}_i|$ *for each agent* i, *and constraints, ii) be permutation invariant with respect to the ordering of individual constraint and neighbor embeddings, and iii) be permutation equivariant with respect to the ordering of agents.*

1. *Problem contexts with any arbitrary number of agents, neighbors, and constraints can be tokenized by the feature embedder (44). Subsequently, the multi-head attention layer, followed by the row-wise feedforward layer, handles an arbitrary number of tokens in the encoder (45) and decoder (48) layers.*

2. *The final permutation invariant transformer model (55) is permutation invariant with respect to the ordering of input tokens by Lemma 3.*

3. *The final permutation invariant transformer model (55) is permutation equivariant with respect to the ordering of agents by Remark 2.*

Suppose that there exist continuous optimal control policies $\{\pi_i^{policy}\}_{i\in[M]}$ *in Problem 2, then invoking (Cybenko, 1989, Theorem 5) concludes the proof.*

I Experiment Details

I.1 Dynamics

Unicycle system In the unicycle system, the state $\boldsymbol{x}$ is defined as $\boldsymbol{x} = [x^{\text{pos}}, y^{\text{pos}}, \theta]^\top \in \mathbb{R}^3$, where x^{pos} is the x-position, y^{pos} is the y-position, and θ is a steering angle. The input $\boldsymbol{u}$ is defined as $\boldsymbol{u} = [v, \omega]^\top \in \mathbb{R}^2$, where v is the velocity, and ω is a turning rate. The unicycle system is described by the following dynamics:

$$\dot{\boldsymbol{x}} = [v \cdot \cos(\theta), v \cdot \sin(\theta), \omega]^\top. \tag{57}$$

Quadrotor system In the quadrotor system, the state $\boldsymbol{x}$ is defined as $\boldsymbol{x} = [x^{\text{pos}}, y^{\text{pos}}, z^{\text{pos}}, v^x, v^y, v^z, \phi, \theta, \psi, p, q, r]^\top \in \mathbb{R}^{12}$, where x^{pos} is the x-position, y^{pos} is the y-position, z^{pos} is the z-position, v^x is the linear x-velocity, v^y is the linear y-velocity, v^z is the linear z-velocity, (ϕ, θ, ψ) are Euler angles (roll, pitch, yaw), and (p, q, r) are body angular rates. In particular, (v^x, v^y, v^z) are velocities in the inertial frame. The input $\boldsymbol{u}$ is defined as $\boldsymbol{u} = [f^{\text{thrust}}, p^{\text{cmd}}, q^{\text{cmd}}, r^{\text{cmd}}]^\top \in \mathbb{R}^4$, where f^{thrust} is the total thrust force (N), and $(p^{\text{cmd}}, q^{\text{cmd}}, r^{\text{cmd}})$ are desired body rates (rad/s). The quadrotor system is described by the following dynamics (Luukkonen, 2011).

$$\dot{\boldsymbol{x}} = \begin{bmatrix} v^x \\ v^y \\ v^z \\ [\cos(\phi)\cdot\sin(\theta)\cdot\cos(\psi) + \sin(\phi)\cdot\sin(\psi)]\cdot f^{\text{thrust}}/m - k_d \cdot v^x/m \\ [\cos(\phi)\cdot\sin(\theta)\cdot\sin(\psi) - \sin(\phi)\cdot\cos(\psi)]\cdot f^{\text{thrust}}/m - k_d \cdot v^y/m \\ -g + (\cos(\phi)\cdot\cos(\theta))\cdot f^{\text{thrust}}/m - k_d \cdot v^z/m \\ p + r\cdot\cos(\phi)\cdot\tan(\theta) + q\cdot\sin(\phi)\cdot\tan(\theta) \\ q\cdot\cos(\phi) - r\cdot\sin(\phi) \\ r\cdot\cos(\phi)/\cos(\theta) + q\cdot\sin(\phi)/\cos(\theta) \\ K_p\cdot(p^{\text{cmd}} - p) \\ K_q\cdot(q^{\text{cmd}} - q) \\ K_r\cdot(r^{\text{cmd}} - r) \end{bmatrix}, \tag{58}$$

where the mass of the quadrotor is $m = 0.83$, the gravity constant is $g = 9.81$, the roll rate gain $K_p = 3.5$, the pitch rate gain $K_q = 3.5$, the yaw rate gain $K_r = 0.3$, and the aerodynamic drag coefficient $k_d = 0.25$.

I.2 Cost functions

Throughout the experiments, we use the quadratic cost function for MPD-DDP defined as

$$J_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \coloneqq \underbrace{\|\boldsymbol{x}_{i,T} - \boldsymbol{x}_i^{\text{ref}}\|^2_{Q_i^f}}_{=\Phi_i(\boldsymbol{x}_{i,T})} + \sum_{k=0}^{T-1} \underbrace{[\|\boldsymbol{x}_{i,k} - \boldsymbol{x}_i^{\text{ref}}\|^2_{Q_i} + \|\boldsymbol{u}_{i,k}\|^2_{R_i}]}_{=l_i(\boldsymbol{x}_{i,k}, \boldsymbol{u}_{i,k})}, \tag{59}$$

where $\boldsymbol{x}_i^{\text{ref}}$ is the target state, Q_i is the state cost matrix, R_i is the control cost matrix, and Q_i^f is the terminal cost matrix for each agent i.

We use the cost function for the multi-agent predictive safety filter is defined as

$$l_i^{\text{safe}}(\boldsymbol{u}_{\theta,i,k}, \tilde{\boldsymbol{u}}_{i,k}) \coloneqq \|\boldsymbol{u}_{\theta,i,k} - \tilde{\boldsymbol{u}}_{i,k}\|_2^2, \qquad \Phi_i^{\text{safe}}(\tilde{\boldsymbol{x}}_{i,T}) = \|\tilde{\boldsymbol{x}}_{i,k} - \boldsymbol{x}_i^{\text{ref}}\|^2_{w^f}, \tag{60}$$

where w^f is the weight coefficient for the terminal cost of the safety filter.

Unicycle system In the experiment in Fig. 2, MPD-DDP uses $Q_i = 15 \cdot \mathrm{diag}([1, 1, 0])$, $R_i = 500 \cdot \mathrm{diag}([1, 0.1])$, $Q_i^f = 10000 \cdot \mathrm{diag}([1, 1, 0.1])$, and the learning-augmented policy uses $Q_i = 15 \cdot \mathrm{diag}([1, 1, 0])$, $R_i = 100 \cdot \mathrm{diag}([1, 0.1])$, $Q_i^f = 10000 \cdot \mathrm{diag}([1, 1, 0.1])$, $w_f = \mathrm{diag}([0, 0, 0])$, for each agent i. In the experiment in Table 2, MPD-DDP uses $Q_i = 15 \cdot \mathrm{diag}([1, 1, 0])$, $R_i = 100 \cdot \mathrm{diag}([1, 0.1])$, $Q_i^f = 10000 \cdot \mathrm{diag}([1, 1, 0.1])$, and the learning-augmented policy uses $Q_i = 15 \cdot \mathrm{diag}([1, 1, 0])$, $R_i = 100 \cdot \mathrm{diag}([1, 0.1])$, $Q_i^f = 10000 \cdot \mathrm{diag}([1, 1, 0.1])$, $w^f = \mathrm{diag}([0, 0, 0])$, for each agent i.

Quadrotor system In the experiment in Table 1, MPD-DDP uses

$$\begin{aligned} Q_i &= \mathrm{diag}([5.0, 5.0, 40, 1.25, 1.25, 1.25, 2.5, 2.5, 0.025, 2.5, 2.5, 2.5]), \\ R_i &= 0.1 \cdot \mathrm{diag}([1.0, 256.0, 256.0, 256.0]), \\ Q_i^f &= 11000 \cdot Q_i, \end{aligned}$$

and the learning-augmented policy uses

$$\begin{aligned} Q_i &= \mathrm{diag}([5.0, 5.0, 10, 1.25, 1.25, 1.25, 2.5, 2.5, 0.025, 2.5, 2.5, 2.5]), \\ R_i &= 0.1 \cdot \mathrm{diag}([1.0, 256.0, 256.0, 256.0]), \\ Q_i^f &= 10000 \cdot Q_i, \\ w^f &= \mathrm{diag}([0.025, 0.025, 0.042, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), \end{aligned}$$

for the nominal swapping task, and $w^f = \mathrm{diag}([0.0313, 0.0313, 0.042, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])$ for the agent failure scenarios, for each agent i.

I.3 Constraints

Unicycle system The maximum wheel speed is constrained as $v_{\text{wheel, max}} = 12.5$ rad/s. The maximum wheel speed constraint is translated to the following linear control constraint:

$$-d_{\max} \leq \frac{1}{2r_{\text{wheel}}} \begin{bmatrix} 2 & l_{\text{axle}} \\ 2 & -l_{\text{axle}} \end{bmatrix} \boldsymbol{u}_i \leq d_{\max}, \quad \forall i \in [N],$$

where $d_{\max} = [v_{\text{wheel, max}}, v_{\text{wheel, max}}]^\top$, $r_{\text{wheel}} = 0.016$m is a wheel radius, $l_{\text{axle}} = 0.11$m is an axle length of a robot. In optimization, we set $v_{\text{wheel, max}} = 12$ rad/s to allow additional margins. The obstacle avoidance constraints are defined as

$$\|\boldsymbol{p}_{i,k} - \boldsymbol{p}_o\|_2 \geq r_o + d_o, \quad \forall o \in \mathcal{O}, \forall i \in [N], \forall k, \tag{61}$$

where $\boldsymbol{p}_{i,k} = [x_{i,k}^{\text{pos}}, y_{i,k}^{\text{pos}}]^\top$ is the position of each agent, $\boldsymbol{p}_o, r_o$ are the position and radius of each obstacle in the set of all obstacles $o \in \mathcal{O}$, and d_o is the desired distance from the obstacle during optimization, where we use $d_o = 0.2$ in Table 2. In Fig. 2, we use $d_o = 0.2$ for all cases, except for the learning-augmented policy at $(20, 30)$-agent settings, where we use $d_o = 0.25$. During the test time, we set $d_o^{\text{test}} = r_i$, where r_i is the radius of each agent i to detect collisions with obstacles.

The inter-agent collision avoidance for each pair of agents in the neighborhood is defined as

$$\|\boldsymbol{p}_{i,k} - \boldsymbol{p}_{j,k}\|_2 \geq d_{\text{safe}}, \quad \forall j \in \mathcal{N}_i \setminus \{i\}, \forall i \in [M], \forall k = 0, \ldots, T, \tag{62}$$

where d_{safe} is the desired safe distance between agents. We set $d_{\text{safe}} = 0.35$ in the simulations in Fig. 2 and Table 2.

Quadrotor system We use the following control constraints:

$$\begin{aligned} 0 \leq f^{\text{thrust}} \leq 26 \quad & \text{(N)} \quad && \text{(total thrust force)} \\ -3 \leq p^{\text{cmd}} \leq 3 \quad & \text{(rad/s)} \quad && \text{(roll rate command)} \\ -3 \leq q^{\text{cmd}} \leq 3 \quad & \text{(rad/s)} \quad && \text{(pitch rate command)} \\ -1 \leq r^{\text{cmd}} \leq 1 \quad & \text{(rad/s)} \quad && \text{(yaw rate command)} \end{aligned}$$

Let $\boldsymbol{p}_{i,k} = [x_{i,k}^{\text{pos}}, y_{i,k}^{\text{pos}}, z_{i,k}^{\text{pos}}]^\top$ be the position of each agent in 3D systems. We use $d_o = 0.4$ for MPD-DDP and $d_o = 0.33$ for the learning-augmented policy in the simulations in Table 1. For inter-agent collision avoidance, we use $d_{\text{safe}} = 1.1$ for MPD-DDP and $d_{\text{safe}} = 1.2$ for the learning augmented policy in Table 1.

I.4 Algorithm configurations

In the experiment in Fig. 2, we commonly use the look-ahead horizon $T = 150$, the number of MPC steps $N^{\text{plan}} = 200$, 1 DDP iteration. The safety filter look-ahead horizon for $(8, 12, 15, 20, 30)$-agent settings is $(7, 7, 10, 7, 7)$, and the threshold ϵ for the state constraint residual gate is $(0.3, 0.44, 0.35, 0.3, 0.25)$ for $(8, 12, 15, 20, 30)$ agents. We replan every 4 time steps at discrete time interval $dt = 0.033$s.

In Table 2, the maximum number of ADMM iterations is 5, and the DDP iterations is 1. The look-ahead horizon is $T = 150$, and the number of MPC steps is 600. We replan every 2 steps.

In Table 1, the look-ahead horizon is $T = 100$ and we replan every time step. The discrete time interval is $dt = 0.05$s. The maximum ADMM iterations for MPD-DDP is 50 and for the learning-augmented policy is 5. The threshold ϵ for the state constraint residual gate is 0.3 in the learning-augmented policy.

I.5 How do merged and nested architectures in MPD-DDP compare in scalability?

Architecture	M	# Collision ↓	Reach ↑	Avg. solve time (s) ↓	Total time (s) ↓
Nested	2	0	2/2	0.023	5.65
	4	0	4/4	0.100	24.10
	8	0	4/8	0.934	224.47
Merged	2	0	2/2	0.027	6.68
	4	0	4/4	0.047	11.54
	8	0	8/8	0.131	32.04

Table 3: Scalability comparison between the nested and merged architectures in MPD-DDP on the unicycle system with a static obstacle for $M \in \{2, 4, 8\}$ agents. The prediction horizon is $T = 150$. Reach refers to the number of target-reached agents out of total number of agents. The number of replanning steps is $N^{\text{plan}} = 240$. We replan every 0.165 s at time interval $dt = 0.033$ s.

In this part, we investigate the scalability of MPD-DDP in a unicycle system with a single static obstacle. The merged and nested architectures refer to two architectures of ADMM-DDP introduced in Saravanos et al. (2023). In Table 3, both architectures show zero collisions across all $M \in \{2, 4, 8\}$. The nested architecture shows diverging behavior in the $M = 8$ setting, resulting, where four out of eight agents do not reach the target. The target reaching condition is described as $\|\boldsymbol{x}_i^{\text{ref}} - \boldsymbol{x}_{i,T^{\text{total}}}\|_2 \leq 0.15$, where $\boldsymbol{x}_i^{\text{ref}}$ is the target state and $\boldsymbol{x}_{i,T^{\text{total}}}$ is the terminal state in the entire horizon for each agent i. Throughout all $(2, 4, 8)$-agent settings, the merged architecture constantly satisfies the time budget of 165 ms, while the nested architecture fails to compute within the time budget in the 8-agent setting on average. Specifically, the merged architecture shows $2.13\times$ faster computation time than the nested architecture, *i.e.*, 47 ms vs. 100 ms per replanning step on average in the 4-agent setting. In the 8-agent setting, the nested architecture shows a steep slowdown from 100 ms (4-agent) to 934 ms, which is $5.66\times$ over time budget. To provide a reproducible result, this simulation experiment assumes a zero-delay setup. The computation time results are based on the average of five repetitions under the same setting to reduce variance.

I.6 What is the effect of dynamic neighborhood set reconstruction in time-varying problem structure?

In this part, we will investigate the impact of the neighborhood set composition. In Fig. 6, trajectory optimization and MPC with a static neighborhood keep the same neighbor set throughout the entire horizon. Subsequently, the neighborhood set does not capture the true distance-based neighbors as the neighbors from the previous time steps start to bifurcate. In contrast, MPC with a dynamic neighborhood recomputes the neighborhood set for each agent at every replanning step, and the neighborhood set illustrated by the ellipse captures the true distance-based neighbors.

In Fig. 5, the number of inter-agent collisions increase as the number of agents M grows in trajectory optimization and MPC with a static neighborhood, *i.e.*, from zero collisions in $M = 4$ to six

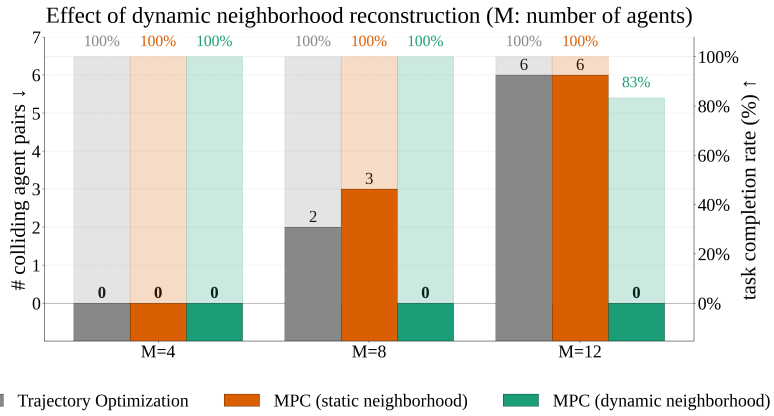


Figure 5: Task completion rate and the number of collisions across $M \in \{4, 8, 12\}$ agents. Solid bars show the number of inter-agent collisions. Transparent overlay shows task completion rate. All settings show zero obstacle collision. All runs use the merged architecture.

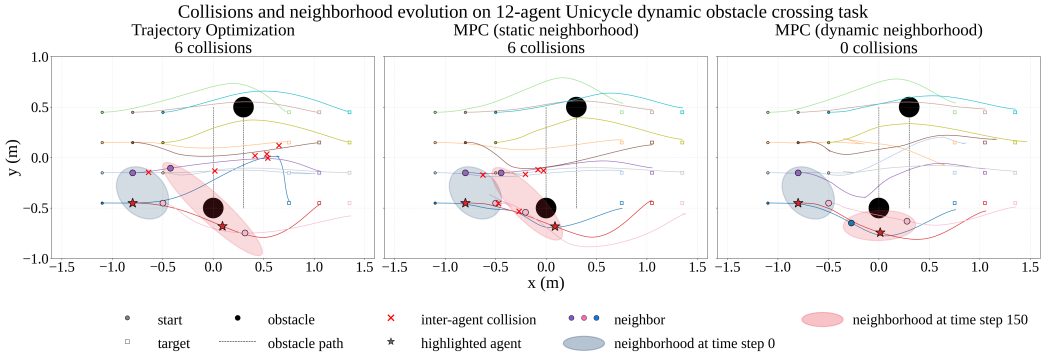


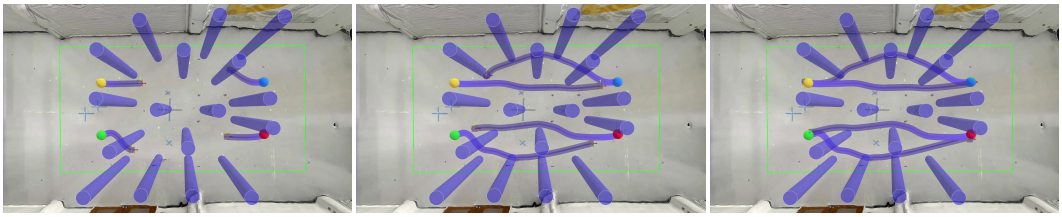
Figure 6: Closed-loop trajectories of $M = 12$ agents. Trajectory optimization and MPC with a static neighborhood show six inter-agent collisions (×), while MPC with dynamic neighborhood reconstruction shows zero collision. Two obstacles are moving up and down, and the static obstacle positions in the figure is an initial state snapshot. The ellipses enclose one highlighted agent and its two neighbors at time instants 0 and 150. Trajectory optimization is solved over the full horizon ($T^{\text{total}} = 1200$), and MPC uses total 240 replanning steps with 165 ms time budget at $dt = 0.33$ s.

colliding pairs in $M = 12$. Notably, MPC with a dynamic neighborhood achieves zero collisions at every M. No obstacle collision is observed in any setting in this experiment.

I.7 Is the proposed method robust against agent failure scenarios?

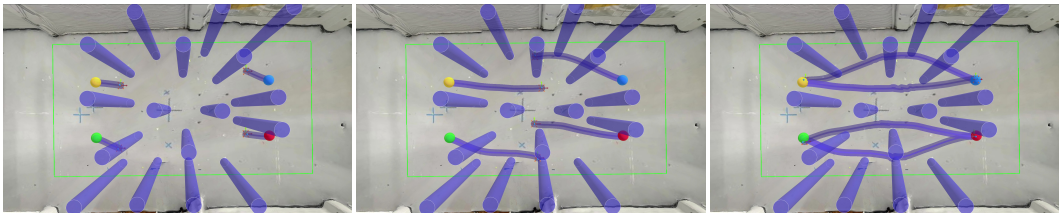
We consider a four-drone position swapping task in a forest-like obstacle field, where a portion of the agents stops operating and lands in the middle of the deployment. We test our methods on the gazebo simulator (Koenig & Howard, 2004) using PX4 software-in-the-loop (SITL) (Meier & The PX4 Contributors) enabled with physical sensor data emulation and Extended Kalman Filter (EKF) state estimation in real time MPC, where the computation time in each replanning step impacts the final result. The algorithms are tested using the position-yaw control scheme. We zero out yaw values to provide in-distribution input to the neural network model. To simulate the agent failure scenario, we insert a landing command on one of the four agents when the traveled distance of the agent exceeds half of the shortest distance from the initial position to the target position, and the optimization algorithms are unaware of the agent failure situation. In this experiment, we do not include the computation time in the first MPC step, as it is dominated by the one-time program compilation.

Maximum number of ADMM iterations for the (MPD-DDP, learning-augmented policy) is (50, 5). The total run time limit for each setting is 120 s for a fair comparison.



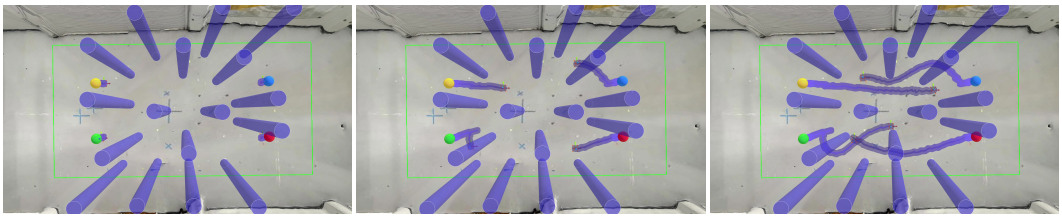
(a) Learning-augmented at 4 s (b) Learning-augmented at 16 s (c) Learning-augmented at 23 s

Figure 7: Learning-augmented policy in the nominal 4-agent swapping scenario: Task finishes within 23 s without collisions.



(a) MPD-DDP at 4 s (b) MPD-DDP at 16 s (c) MPD-DDP at 42 s

Figure 8: MPD-DDP in the nominal 4-agent swapping scenario: Task finishes within 42 s without collisions.



(a) Centralized SQP at 4 s (b) Centralized SQP at 42 s (c) Centralized SQP at 120 s

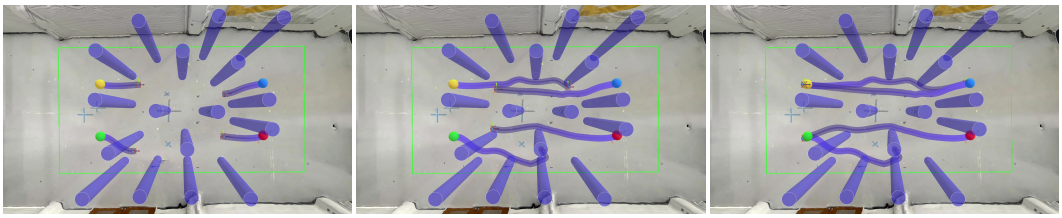
Figure 9: Centralized SQP in the nominal 4-agent swapping scenario: Task does not finish within 2 minute time limit.

I.8 What is the effect of the neural network architecture in solving multi-agent control problems?

We use i) a sequence-to-sequence transformer model with positional encoding, ii) sum-pooling for the set readout function in Deep sets, iii) Gated Recurrent Unit (Cho et al., 2014) and teacher-forcing training (Williams & Zipser, 1989) for RNN. In Table 4, we report the off-policy evaluation of the neural network architectures in Average Displacement Error (ADE) and Final Displacement Error (FDE) with respect to the expert policy (MPD-DDP) on 99 obstacle-field navigation test problems with 8 unicycle agents and 6 obstacles in randomized positions and sizes. We evaluate ADE and FDE on the same initial state from the same rollout trajectories of MPD-DDP. Permutation invariant Transformer model shows the smallest ADE and FDE values among other neural network architectures, and the performance remains invariant even when the order of the input tokens is reversed. The performance of Transformer and RNN collapses in the case of reversed token ordering.

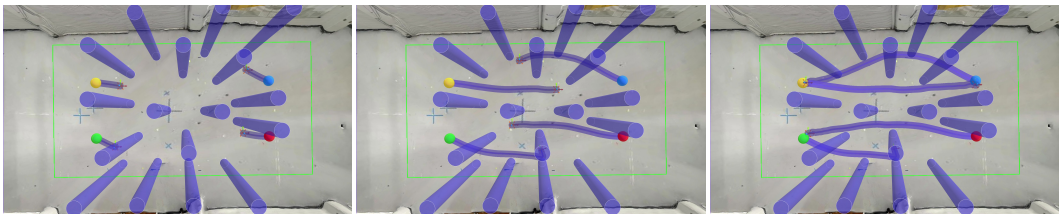
I.9 Hardware experiments

Fig. 13 illustrates the trajectories of the learning-augmented policy in the 8-to-8 and 8-to-12 agent problem transfer experiment on the Robotarium platform. To prevent physical collisions between robots, control barrier function (Ames et al., 2019) has been additionally applied at the platform



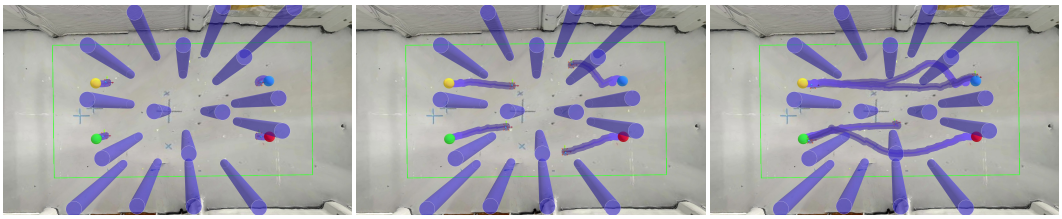
(a) Learning-augmented at 4 s (b) Learning-augmented at 16 s (c) Learning-augmented at 21 s

Figure 10: Learning-augmented policy in the 4-agent swapping with agent failure scenario: Task finishes within 21 s without collisions.



(a) MPD-DDP at 4 s (b) MPD-DDP at 16 s (c) MPD-DDP at 60 s

Figure 11: MPD-DDP in the 4-agent swapping with agent failure scenario: Task finishes within 60 s without collisions.



(a) Centralized SQP at 4 s (b) Centralized SQP at 42 s (c) Centralized SQP at 120 s

Figure 12: Centralized SQP in the nominal 4-agent swapping scenario: Task does not finish within 2 minute time limit.

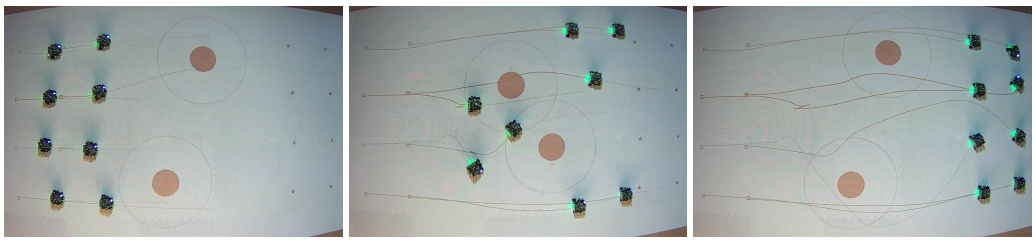
level. The proposed method successfully finishes the task in 8-to-8 and 8-to-12 experiments throughout three repetitions, *i.e.*, all agents reached their target positions within 0.15 m with zero collisions.

In the 8-to-8 transfer experiment, the (average, median, maximum) solve times are recorded $(0.118, 0.091, 0.384)$ s, and the (average, maximum) numbers of ADMM iterations are $(1.42, 3)$. From a total of 487 solves, the safety verifier accepted 78.9% of the learned trajectories.

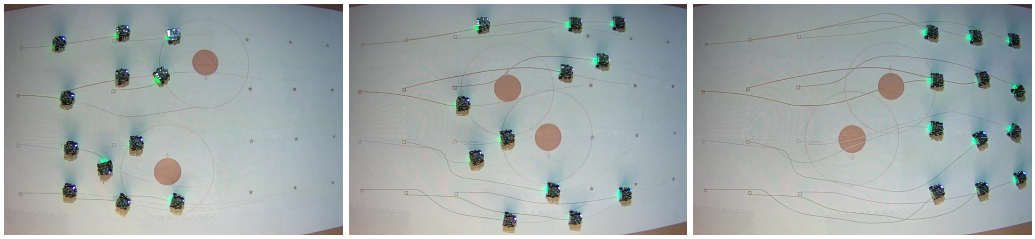
In the 8-to-12 transfer experiment, the (average, median, maximum) solve times are recorded $(0.163, 0.118, 0.454)$ s, and the (average, maximum) numbers of ADMM iterations are $(1.65, 3)$. From a total of 330 solves, the safety verifier accepted 67.6% of the learned trajectories.

Model	Ordering	ADE (m) ↓	FDE (m) ↓
Permutation invariant transformer (Lee et al., 2019)	nominal	0.0196	0.0282
	reversed	0.0196	0.0282
Transformer (Vaswani et al., 2017)	nominal	0.0206	0.0304
	reversed	0.1198	0.2573
Deep Sets (Zaheer et al., 2017)	nominal	0.1765	0.3497
	reversed	0.1765	0.3497
RNN (Cho et al., 2014)	nominal	0.0646	0.1611
	reversed	0.4914	1.0019

Table 4: Permutation invariant Transformer outperforms the other architectures in Average Displacement Error (ADE) and Final Displacement Error (FDE) with respect to the expert policy (MPD-DDP) in the nominal case and the performance remains invariant in the reversed ordering case. The vanilla Transformer scores ADE and FDE close to the permutation invariant Transformer in the nominal case, but collapses in the reversed ordering setting. We report off-policy evaluation result of the learned policy at the same reference state trajectories generated by the expert (MPD-DDP) on 99 obstacle-field navigation test problems with 8 unicycle agents and 6 obstacles in randomized positions and sizes. ADE (Average Displacement Error) and FDE (Final Displacement Error) are evaluated with respect to the expert plan from the same state, evaluated on 40 rollouts, every 20th replanning step. ADE is the average of the displacement error of the entire 150-step plan in the predicted trajectory. FDE is the displacement error of the terminal state in the predicted trajectory. Reversed ordering changes the input block order from (self, obstacle, neighbor) to (neighbor, obstacle, self) and reverses the order of the input tokens within each block.



(a) 8-to-8 agent transfer at 3 s (b) 8-to-8 agent transfer at 10 s (c) 8-to-8 agent transfer at 29 s



(d) 8-to-12 agent transfer at 3 s (e) 8-to-12 agent transfer at 7 s (f) 8-to-12 agent transfer at 20 s

Figure 13: Real-world experiments for testing transferability of the learned model with safety filter. The policy trained on 8-agent setting is deployed to: 8-agent setting in (a)-(c), 12-agent setting in (d)-(f). Two red circles in the center represents moving obstacle. All agents complete the task within (29 s, 20 s) in (8-to-8, 8-to-12) agent transfer experiments without any collisions.