DECENTRALIZED MASTER-MIND: JOINT ACTION REFINEMENT THROUGH ITERATIVE INTENT DENOISING IN MULTI-AGENT PATHFINDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Decentralized multi-agent path finding (MAPF) with communication requires agents to reach individual goals without collisions under partial observability. Learnable policies trained on expert data provide an effective approach to this problem. However, when several coordinated joint actions are valid in the same context, independently sampling from per-agent distributions can recombine locally valid choices into incompatible joint actions. This failure can arise from the final sampling mechanism even when the per-agent action distributions are learned correctly. **DMM**(Decentralized Master-Mind) addresses this by replacing one-shot action sampling with discrete, iterative refinement of action intents across communication rounds, inspired by iterative refinement in generative models such as diffusion and flow matching. Agents initialize random action intents and refine them through local communication, coupling their choices before commitment. DMM is pretrained with imitation learning on expert MAPF solutions and further optimized with **MICPO**, a critic-free group-relative reinforcement-learning method designed for multi-agent, multi-round action refinement. DMM generally achieves higher success rates and lower solution costs than the evaluated learnable baselines. On 1,600 MovingAI tasks, DMM fine-tuned with MICPO solves 1,598, the highest coverage among the evaluated methods, while achieving solution costs close to those of the strongest baselines. DMM also scales to **over one million** simultaneously acting agents in obstacle-rich environments. These results show that round-level intent refinement can improve joint-action coordination while preserving decentralized execution.

1 INTRODUCTION

Multi-Agent Path Finding (MAPF) asks agents to reach individual goals without collisions (Stern et al., 2019). Centralized search faces a combinatorial joint state space (Surynek, 2010; Sharon et al., 2015), motivating learned policies that act from local observations in parallel. Reinforcement learning and imitation learning have produced effective decentralized policies (Phan et al., 2024; Andreychuk et al., 2025b; Veerapaneni et al., 2025); learned communication lets agents condition their decisions on neighboring agents (Ma et al., 2021; Jain et al., 2026a). Yet local decisions can still fail to coordinate when several joint actions are valid.

Despite architectural differences, these policies commonly sample each agent's final action independently. If the joint-action distribution has multiple valid modes, combining locally plausible actions can yield an invalid joint action. In the corridor of Figure 1, two coordinated resolutions are valid, but the other two action combinations stall or collide. Such mismatches are especially costly in reactive execution: after a joint action is applied, the policy cannot discard it and branch from an earlier state as a search method can. Communication enriches each agent's context; independent sampling after the last communication round still cannot couple the agents' remaining choices. We formalize this as a decentralized factorization gap: even Bayes-optimal per-agent predictions need not recover the expert joint-action distribution. An execution-time shield can enforce feasibility without resolving this independent-sampling limitation in the learned policy itself. Agent-wise action accuracy is therefore an incomplete measure of coordination; the combined decisions must be judged by whether the team reaches its goals and at what cost.

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.

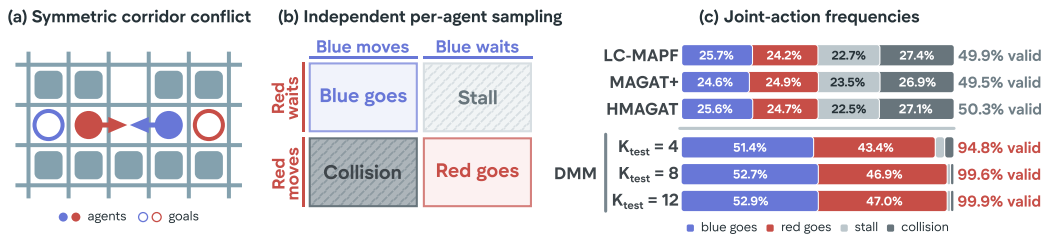

Figure 1: Symmetric corridor conflict and joint-action frequencies (1,000 samples per checkpoint, averaged over five checkpoints). Independently sampled LC-MAPF, MAGAT+, and HMAGAT actions yield about 50% valid joint actions. DMM trained at $K_{\text{train}} = 4$ reaches 94.8%, 99.6%, and 99.9% at $K_{\text{test}} = 4, 8, 12$.

We introduce **Decentralized Master-Mind (DMM)**, which refines stochastic action intents through several communication rounds before committing an action. Exchanging evolving intents lets agents influence one another's choices while preserving decentralized policy execution. We pretrain DMM by imitation and improve complete rollouts with **Multi-agent Iterative Commitment Policy Optimization (MICPO)**, a critic-free method that compares matched trajectories and optimizes sampled decisions at each refinement round (Shao et al., 2024).

In the POGEMA standalone-policy benchmark, DMM-MICPO-3M achieves the highest success rate at the largest team size and the lowest median SoC ratio in every domain. On MovingAI, it combines established CS–PIBT shielding with our Repeat-State Escape (RSE), which seeks an alternative successor when the proposed joint configuration has been visited before, and solves 1,598 of 1,600 tasks. The smaller DMM-MICPO-0.8M has the lowest median runtime in the MovingAI comparison and achieves the first full solutions of obstacle-rich million-agent MAPF instances by a learning-based policy.

2 Related Work

Classical MAPF spans optimal search and reductions (Sharon et al., 2015; Surynek et al., 2016), reactive coordination with PIBT (Okumura et al., 2022), and search or repair methods built around local planning. PIBT resolves immediate conflicts by priority inheritance and backtracking without exploring a search tree. LaCAM uses such coordinated moves to generate successors in a search over configurations (Okumura, 2023); MAPF-LNS2 repairs subsets of paths in a candidate solution (Li et al., 2022). These different mechanisms trade solution quality and guarantees against computation, and provide complementary baselines for learned policies.

Learned decentralized policies range from direct local prediction, as in PRIMAL and MAPF-GPT (Sartoretti et al., 2019; Andreychuk et al., 2025b), to graph-based communication in MAGAT and HMAGAT (Li et al., 2021; Jain et al., 2026a) and multi-round Transformer communication in LC-MAPF (Vyaltsev et al., 2026). Communication enriches each agent's context, but sampling a final action independently after communication ends can still recombine incompatible choices from different valid joint modes. DMM instead exchanges *intermediate action intents* while those choices are being refined.

Execution-time correlation can also be obtained by choosing agents' actions autoregressively and revealing earlier choices to later agents (Fu et al., 2022). This imposes an action order; DMM retains simultaneous commitment after local communication. DiffLNS (Wang et al., 2026) also uses iterative refinement, but centrally constructs a full-horizon joint plan rather than a decentralized action at each timestep. Further distinctions, including continuous-space multi-robot refinement, are detailed in Appendix A.

3 Problem Setup

A MAPF instance specifies a four-connected grid $G = (V, E)$ and starts s_u and goals g_u for agents $u \in U$, with $n = |U|$. At each timestep, every agent selects an action from $\mathcal{A} =$

$\{\text{wait}, \text{up}, \text{down}, \text{left}, \text{right}\}$. A joint action is feasible if moves follow grid edges and avoid both vertex collisions and opposing traversals of the same edge. An instance is solved when all agents reach their goals within horizon H. If c_u is the first timestep at which agent u reaches its goal and stays there, solution quality is measured by

$$\text{SoC} = \sum_{u \in U} c_u, \qquad \text{MS} = \max_{u \in U} c_u. \tag{1}$$

Success rate is the fraction of instances solved within H.

We study shared-parameter decentralized policies. Each agent observes a local egocentric grid patch and communicates with nearby agents over K synchronous rounds before all agents commit to actions simultaneously. Messages may depend on private randomness, as in DMM. The learned policy's action proposal is decentralized: an agent uses only its observation and received local messages, does not see other agents' committed actions before choosing its own, and has no access to the global state or an ordering over agents. The POGEMA policy comparison executes these proposals without shielding, whereas the MovingAI DMM configuration applies centralized CS–PIBT and RSE before execution. DMM is pretrained from expert actions and then optimized on team-level rollout outcomes; the formal Dec-POMDP and training-objective definitions are given in Appendix B.

4 Method

The Decentralized Factorization Gap

Communication enriches each agent's context, but independent final sampling still discards correlations between agents' actions. Thus, even exact per-agent imitation need not reproduce the expert joint distribution.

Let $P^\star$ denote the expert distribution. Let $A = (A_1, \ldots, A_n)$ be the expert joint action and $X = (X_1, \ldots, X_n)$ the information available at action commitment, where X_u is the local information available to agent u and X_{-u} denotes the remaining context. For an independently sampling decentralized executor, let $q_u(a_u \mid x_u)$ denote the action distribution used by agent u given its local information. Its joint distribution has the product form

$$Q_q(a \mid x) = \prod_{u \in U} q_u(a_u \mid x_u). \tag{2}$$

Residual expert-action dependence given the full available context is measured by conditional total correlation (Watanabe, 1960):

$$\text{TC}(A \mid X) := \mathbb{E}_X D_{\text{KL}}\left(P^\star(A \mid X) \,\middle\|\, \prod_{u \in U} P^\star(A_u \mid X) \right). \tag{3}$$

Proposition (decentralized factorization gap). Even with arbitrarily expressive local policies, the best product executor satisfies

$$\begin{aligned} \mathcal{G}_{\text{dec}} &:= \min_{\{q_u\}_{u \in U}} \mathbb{E}_X D_{\text{KL}}(P^\star(A \mid X) \,\|\, Q_q(A \mid X)) \\ &= \text{TC}(A \mid X) + \sum_{u \in U} I(A_u; X_{-u} \mid X_u) \\ &\geq \text{TC}(A \mid X). \end{aligned} \tag{4}$$

The minimum is attained by the Bayes-optimal per-agent cross-entropy policies $q_u^\star(A_u \mid X_u) = P^\star(A_u \mid X_u)$. The two terms respectively measure residual action dependence and action-relevant information missing from an agent's local context. Positive total correlation therefore implies a strictly positive gap. In the symmetric corridor, an expert choosing (g, w) or (w, g) equally often has exact uniform local marginals, yet independent sampling produces a collision or stall half the time. Once a deterministic communication transcript is fixed, final independent draws remain a product distribution. DMM instead exchanges sampled intents before commitment. The proof, corridor calculation, and related bounds are given in Appendix B.3.

DMM: Decentralized Iterative Intent Refinement

DMM augments a decentralized communication policy with an *action intent* that is refined during communication and included in the messages exchanged with neighboring agents. Each agent encodes its local observation into a compact representation ℓ_u, which remains fixed across refinement rounds, while the communicated intent evolves across rounds. The experiments instantiate this mechanism with two network architectures, DMM-3M and DMM-0.8M, described in Appendix C.

The intent update could be trained as a standard refinement step, as in diffusion or flow matching. However, DMM couples refinement with learned communication: each round's state determines the message passed to the next round. This prevents independent training of refinement steps and motivates the sequential communication-refinement process described next.

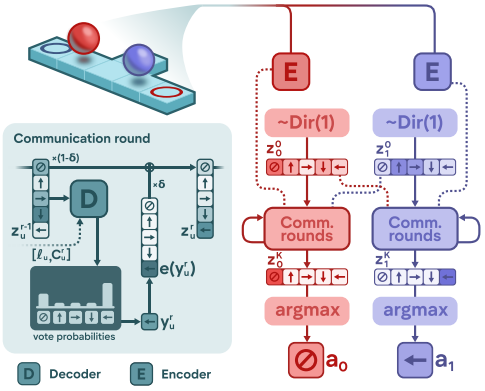

Figure 2: DMM exchanges sampled intents for K rounds before action commitment.

Intent initialization. Agent u maintains an intent $z_u^r \in \mathbb{R}^{|\mathcal{A}|}$, with one value for each action. Rather than representing a probability vector directly, the intent is maintained in mean-centered log-space, which preserves relative action preferences while removing their arbitrary common offset. At inference, the initial intent is sampled independently from an uninformative Dirichlet prior, $z_u^0 = \mathrm{lc}(\log d_u)$ with $d_u \sim \mathrm{Dirichlet}(\mathbf{1})$, where $\mathrm{lc}(v) = v - \bar{v}$ denotes mean-centering.

Iterative refinement. Figure 2 illustrates the iterative communication and refinement process. At each round $r \in \{1, \ldots, K\}$, agent u broadcasts its current intent together with a learned message feature:

$$m_u^r = h_u^{r-1} + W_s z_u^{r-1}, \tag{5}$$

where h_u^{r-1} is the learned message feature and W_s projects the intent into the message space. Agent u then collects the messages of its neighbors and itself, $C_u^r = \{m_v^r\}_{v \in \mathcal{N}(u) \cup \{u\}}$, and the decoder combines them with its fixed latent representation ℓ_u to give $\psi_u^r = \mathrm{Decoder}(\ell_u, C_u^r)$, from which two heads produce the action logits $\phi_u^r = \mathrm{PolicyHead}(\psi_u^r)$ and the next message feature $h_u^r = \mathrm{MsgHead}(\psi_u^r)$. From the action logits, the agent samples a discrete vote $y_u^r \sim \mathrm{Categorical}(p_{\theta,u}^r)$ with $p_{\theta,u}^r = \mathrm{softmax}(\phi_u^r)$, which is then incorporated into the current intent:

$$z_u^r = (1 - \delta) z_u^{r-1} + \delta e(y_u^r), \tag{6}$$

where $e(y) = \mathrm{lc}(\log(\mathrm{onehot}(y) + \eta))$ with $\eta > 0$ a small constant that prevents taking the logarithm of zero. The embedding places the vote in the same centered log-space as the intent. The step size $\delta \in (0, 1)$ retains part of the previous intent while incorporating the current vote, so the intent accumulates information across refinement rounds. Because the vote is discrete, gradients do not pass through the vote or the intent update. The message feature remains differentiable, however, allowing later-round losses to influence earlier communication.

The key idea in DMM is that the quantity being refined is also part of the communication signal. Each agent therefore observes its neighbors' evolving intents and can adapt its own subsequent votes before commitment. Although the initial intents are sampled independently, the refinement trajectories become coupled through communication, allowing different runs to settle on different coordinated joint-action outcomes. After the K refinement rounds, each agent commits to the action favored by its own final intent, $a_u^t = \arg\max_{a \in \mathcal{A}} z_u^K[a]$. Agents therefore coordinate through their exchanged refinement states before commitment, while each agent selects its final action locally from its own refined intent. Unlike independent final-action sampling, this lets neighbors respond to sampled preferences before commitment; the final choice is deterministic given the resulting intent. Any collision shielding used in later evaluations is downstream of this decentralized policy.

IMITATION PRETRAINING

DMM is first pretrained on expert trajectories by supervising the votes at every refinement round. For an expert action $a_u^{t,\star}$, the imitation loss is $\mathcal{L}_{\mathrm{IL}} = -\frac{1}{K}\sum_{r=1}^{K} \log p_{\theta,u}^r(a_u^{t,\star})$.

Early in training, however, noisy model votes produce uninformative intents that are then communicated to neighboring agents. We therefore use two forms of teacher forcing. With probability β_0, the initial intent is biased toward the expert action by sampling $d_u \sim \mathrm{Dirichlet}\left(\mathbf{1} + \mathrm{onehot}(a_u^{t,\star})\right)$ instead of the uninformative prior above. With probability β_r at each round, the sampled vote y_u^r is replaced by the expert action before applying Eq. (6). The first intervention provides an informative initial condition; the second keeps the evolving intent trajectory informative for neighboring agents. The latter is particularly relevant to DMM because the resulting intent is communicated in subsequent rounds, so errors in early votes can also alter the context received by neighboring agents. Both probabilities are annealed during pretraining toward a retained nonzero floor, reducing the amount of teacher forcing while preserving an expert signal in the evolving intent trajectory. The teacher-forcing schedules used in our experiments are specified in Appendix D.2.

Imitation teaches local agreement with coordinated expert actions but does not directly optimize the quality of the resulting joint solution. We therefore fine-tune the same refinement process directly on task-level outcomes.

MICPO: CRITIC-FREE MULTI-AGENT FINE-TUNING

MICPO fine-tunes complete decentralized rollouts with a shared team objective, without a value function. It adapts group-relative optimization (Shao et al., 2024) to DMM's multi-agent, multi-round decisions. For each training scenario, a group of trajectories shares the sampled initial intent z_t^0 at every environment step, but samples subsequent refinement votes independently; their states may diverge after earlier decisions.

Each trajectory ζ receives a team return combining agents' off-goal durations $q_c(\zeta)$ and blocked non-wait proposals $q_b(\zeta)$, with fixed scalar weights $w_c = 1.0$ and $w_b = 0.3$. Because the two components have different scales, they are normalized separately within each matched group and recombined with the same weights, following component-wise group normalization (Liu et al., 2026a):

$$\hat{A}(\zeta) = N_G(w_c N_G(q_c(\zeta)) + w_b N_G(q_b(\zeta)))\,, \tag{7}$$

where N_G standardizes within the group and returns zero when the group standard deviation falls below a threshold τ. This single advantage is shared across all agents and rounds of the rollout. Replay ranks trajectories by their unnormalized weighted return and retains the highest and lowest from each group, with advantages computed from the full group before filtering.

Optimization acts on each sampled vote rather than an episode-level action. For the vote y_u^r, the importance ratio $\rho_u^r = p_{\theta,u}^r(y_u^r)/p_{\mathrm{old},u}^r(y_u^r)$ is computed separately for every agent and refinement round. The full objective averages the clipped term over both and adds a KL penalty to the frozen imitation-pretrained reference policy $p_{\mathrm{ref},u}^r$:

$$\begin{aligned}\mathcal{L} = & -\mathbb{E}_{u,r}\left[\min\left(\rho_u^r \hat{A},\ \mathrm{clip}\left(\rho_u^r, 1-\varepsilon, 1+\varepsilon\right)\hat{A}\right)\right] \\ & + \alpha_{\mathrm{KL}} \mathbb{E}_u\left[\sum_{r=1}^{K} D_{\mathrm{KL}}\left(p_{\theta,u}^r \,\middle\|\, p_{\mathrm{ref},u}^r\right)\right].\end{aligned} \tag{8}$$

All K rounds of a selected timestep are replayed together, recomputing messages so gradients pass through communication, while the reference anchors each round's action distribution and limits drift. Exact rewards, normalization, losses, and replay procedure appear in Appendix D.3.

5 EXPERIMENTS

Our experiments assess DMM as a standalone, unshielded policy on POGEMA (Skrynnik et al., 2025) and as part of a solver with CS–PIBT and our Repeat-State Escape (RSE) on MovingAI (Stern et al., 2019). We also test scalability to 1,048,576 agents and use a controlled corridor conflict to examine joint-action coordination and the effect of teacher forcing on policy quality.

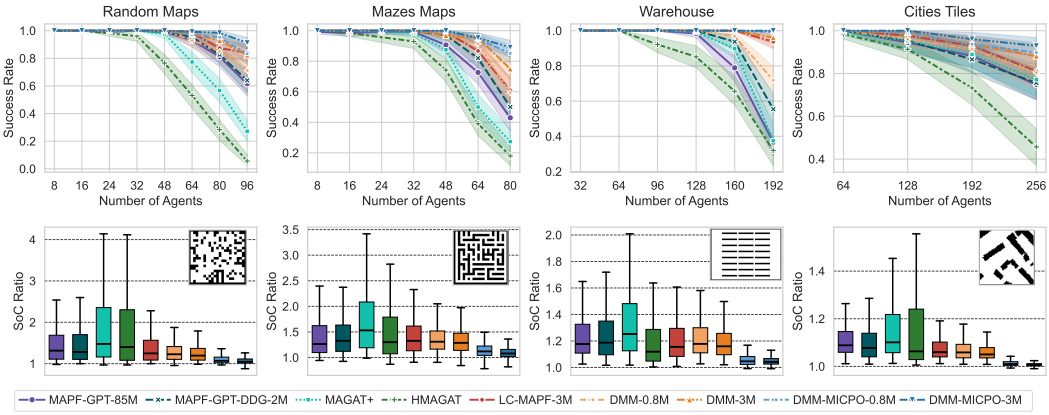

Figure 3: POGEMA results. Top: success rate versus agent count (128 instances per point; shaded 95% Wilson CIs). Bottom: SoC ratio to centralized LaCAM* on solved instances, pooled across all agent counts per map.

DMM configurations and training. Both DMM implementations use the same intent-refinement mechanism. DMM-3M retains the LC-MAPF Transformer encoder–decoder backbone (Vyaltsev et al., 2026); DMM-0.8M reuses observation-dependent computation across rounds and has 763,296 parameters, about $4.25\times$ fewer than DMM-3M. Both are pretrained by imitation for one million iterations, then fine-tuned with MICPO. Their architectures, training data, schedules, and compute are detailed in Appendix C, Appendix D.4, and Appendix D.2.

Evaluation hardware. All evaluations were conducted on a server with two Intel Xeon Platinum 8480+ CPUs (56 cores each, 2.0–3.8 GHz), 2 TB of system memory, and four NVIDIA H100 80 GB GPUs. In all GPU-based evaluations other than the large-scale scalability experiment, each worker used one GPU and processed independent instances. In the large-scale experiment, each rollout was distributed across all four GPUs.

POGEMA BENCHMARK

Following LC-MAPF (Vyaltsev et al., 2026), we compare DMM with HMAGAT (Jain et al., 2026a), MAGAT+ (Jain et al., 2026b), MAPF-GPT-85M (Andreychuk et al., 2025b), MAPF-GPT-DDG-2M (Andreychuk et al., 2025a), and LC-MAPF-3M on POGEMA's *Random*, *Mazes*, *Warehouse*, and *Cities-Tiles* domains. Every method is a standalone learned policy: no shielding, search, or action repair is used. Episodes last at most 128 steps, or 256 on Cities-Tiles; domain sizes and generalization roles are given in Appendix E.1.

Figure 3 compares success and LaCAM*-normalized sum-of-costs (SoC) across four POGEMA domains. At the largest Warehouse and Cities-Tiles team sizes (192 and 256 agents), DMM-MICPO-3M succeeds on 1.000 and 0.922 of tasks, versus 0.938 and 0.805 for LC-MAPF-3M, the strongest baseline there.

DMM-3M and LC-MAPF-3M share an encoder-decoder backbone, communication bottleneck, training data, and parameter count; DMM instead produces actions after K discrete intent-refinement rounds. It generally achieves higher success and lower pooled SoC ratios, especially with larger teams, consistent with a benefit from refinement. Round-count and message-content ablations appear in Appendix E.2 and Appendix E.3. The lighter DMM-0.8M generally matches or exceeds LC-MAPF-3M across the evaluated settings, motivating its use in the scalability study.

MICPO improves both DMM sizes over imitation-only training, especially in large-team success, while shifting SoC ratios downward. Among solved tasks, DMM-MICPO-3M has the lowest median and narrowest interquartile range in every domain (medians 1.006–1.102); MAGAT+ and HMAGAT have the broadest spreads and heaviest upper tails (medians 1.063–1.892).

MovingAI benchmark

We consider every available MovingAI scenario (25 even and 25 random per map) and select its maximum-agent task. We exclude only the 50 tasks from `maze-128-128-1`: none of the evaluated methods solved them in preliminary runs. The resulting 1,600 tasks span 32 of 33 maps and 32–7,070 agents; details are in Appendix F. We compare DMM with search-based LG-LaCAM (Arita & Okumura, 2026) and MAPF-LNS2 (Li et al., 2022), hybrid LaGAT (Jain et al., 2026b), and learned HMAGAT (Jain et al., 2026a). Search-based and hybrid methods receive 600 wall-clock seconds per task; reactive learned policies receive 5,000 environment steps.

Every method stops at its first valid solution or at its budget, with no post-solution refinement. We report success, time to a valid solution, and solution quality: an unsolved task cannot be optimized later, and both the remaining time and the supplied solution quality matter to optional time-limited LNS (Arita & Okumura, 2026), which is outside our evaluation.

The reactive policies use established CS–PIBT collision shielding (Veerapaneni et al., 2025; Jain et al., 2026a). We introduce Repeat-State Escape (RSE) to address a different failure mode: cycles among valid, shielded joint configurations. RSE records visited configurations. If CS–PIBT proposes one again, RSE considers agents in PIBT priority order and temporarily forbids the first unfinished agent's proposed transition for which an alternative remains. It reruns CS–PIBT from the current state to seek a different successor. Constraints accumulate only during these retries and are discarded after an action is executed; RSE neither rolls back nor builds a search tree. Both CS–PIBT and RSE use global state and are centralized.

Figure 4 shows that DMM leads in coverage despite operating reactively: RSE can revise a proposed next state, but cannot branch from an earlier executed state. In the sampling ablation, RSE improves HMAGAT coverage but has little effect on either DMM variant, so it alone does not explain DMM's advantage. We test sampling, sampling with RSE, and argmax with RSE; the main comparison uses sampling with RSE for HMAGAT and argmax with RSE for DMM. The full comparison and implementation details are in Appendix F.1 and Appendix F.

DMM-MICPO-3M matches the per-task virtual-best SoC on 736 tasks (LG-LaCAM: 698), with mean/median/q_{95} ratios of 101.8/100.1/107.3%. It solves all 50 tasks on 31 maps and 48 on the remaining map; its map-wise mean ratio never exceeds 108.9%. LG-LaCAM combines 1,569 solved tasks with a 100.3% median, but high-cost tails on maze, room, and narrow-corridor game maps raise its mean/q_{95} ratios to 121.3/164.0%.

DMM-MICPO-0.8M has the lowest median runtime (2.33 seconds), compared with 2.87 for LaGAT, 5.61 for LG-LaCAM, 8.27 for DMM-MICPO-3M, 8.43 for MAPF-LNS2, and 48.10 for HMAGAT. Thus, the compact model favors speed and the 3M model coverage and quality, with runtime comparable to MAPF-LNS2. These are runtimes of the actual solver configurations, not equalized compute: search/hybrid methods are time-limited, whereas learned policies are step-limited.

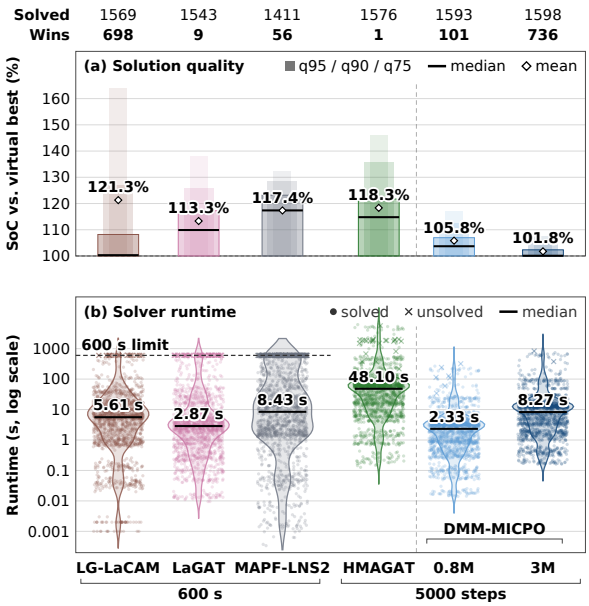

Figure 4: MovingAI-1600. Top: solved tasks and virtual-best SoC wins (ties count). **(a)** SoC ratios to the per-task best of six methods; bars show the 75th/90th/95th percentiles. **(b)** Solver runtime (log scale); dots/crosses denote solved/unsolved tasks.

MILLION-AGENT SCALABILITY

We test whether a learned policy remains effective beyond standard benchmark scales by comparing DMM-MICPO-0.8M with GPU CS–PIBT against GPU-PIBT with BFS-based move preferences. At each of four population sizes (131,072, 262,144, 524,288, and 1,048,576 agents), both methods run on four matched 2304×2304 POGEMA mazes with approximately 29.1% obstacles. Starts and goals are distinct; agent density spans 3.48–27.86% of free cells. The horizon is 32,768 steps, and each rollout uses the four H100 GPUs described above. Unlike an earlier million-agent test on an empty grid with start–goal distances capped at 64 (Andreychuk et al., 2025a), these maze scenarios impose no trip-length cap. The full protocol is in Appendix G.1.

Table 1: Completion on four matched mazes per population size. Density is agents per free cell; on-goal fraction, steps, and total wall are means across four scenarios, with length extrema shown. GPU-PIBT wall time ends at the 32 768-step horizon, not at a solution.

Method	Agents	Density (%)	On goal (%)	Mean steps	Min.	Max.	Total wall (min)
DMM-MICPO-0.8M	1 048 576	27.86	100.00	20 867	16 345	29 508	256.9
GPU-PIBT	1 048 576	27.86	89.15	32 768	32 768	32 768	134.3
DMM-MICPO-0.8M	524 288	13.93	100.00	8 389	7 600	8 989	75.7
GPU-PIBT	524 288	13.93	95.46	32 768	32 768	32 768	21.9
DMM-MICPO-0.8M	262 144	6.97	100.00	5 185	4 961	5 461	43.1
GPU-PIBT	262 144	6.97	97.83	32 768	32 768	32 768	10.1
DMM-MICPO-0.8M	131 072	3.48	100.00	4 704	4 647	4 769	35.0
GPU-PIBT	131 072	3.48	98.91	32 768	32 768	32 768	7.2

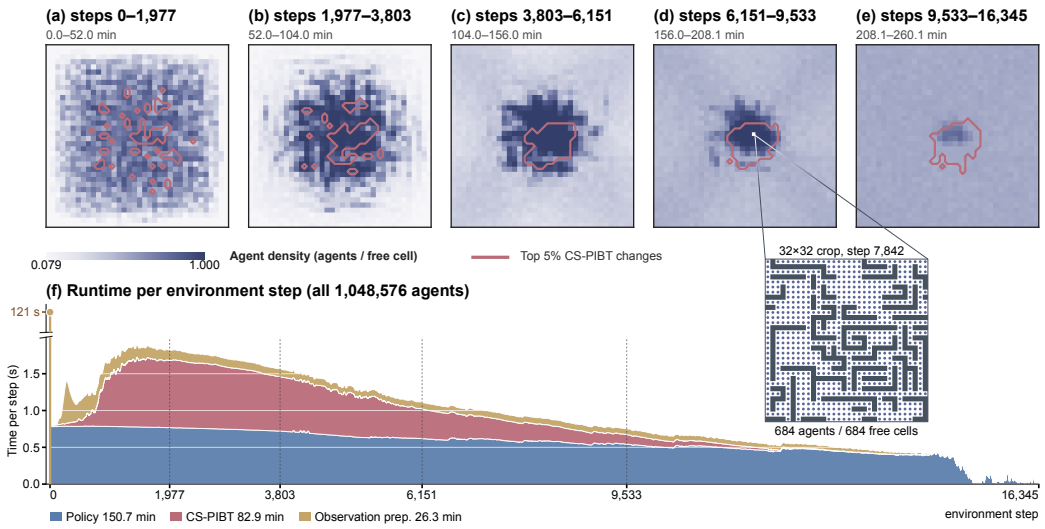

Figure 5: Million-agent DMM rollout (16,345 steps). **(a–e)** Agent density and top-5% CS–PIBT action-change regions over five 52-minute windows. **(f)** Per-step stage time (maximum across four GPUs, 32-step smoothing); broken axis: 121-second BFS prefill.

Table 1 shows that DMM solves all 16 scenarios; GPU-PIBT solves none, despite averaging 89.15% of agents on goal at one million agents. DMM finishes those runs in 16,345–29,508 steps.

At one million agents, DMM averages 763 ms per step, 4.28 hours per completed rollout, and 49.07 GiB peak memory per GPU. GPU-PIBT averages 229 ms per step and reaches the horizon in 2.24 hours without solving. Across the eightfold population increase, mean step time grows $1.73\times$ for DMM versus about $21\times$ for GPU-PIBT. Figure 5 shows where shield interventions concentrate in the crowded center. As traffic clears and agents finish, skipped policy inferences reduce the work per step.

CORRIDOR CONFLICT

In the two-agent corridor of Figure 1, two expert solutions share one ambiguous state where either agent can yield. We train DMM, LC-MAPF, MAGAT+, and HMAGAT on both trajectories for 5,000 iterations and sample 1,000 joint actions per trained seed at that state, across five seeds. We compare DMM's retained teacher-forcing floor of 0.8 with a zero-floor variant ($\beta = \beta_0 = \beta_r$); the full training and sampling protocol is in Appendix H.1.

DMM with the retained teacher-forcing floor $\beta_0 = \beta_r = 0.8$ places 94.8% of its samples on the two valid resolutions (Table 2). In contrast, LC-MAPF, MAGAT+, and HMAGAT place approximately equal probability on all four joint actions, with about 50% of their samples falling on the two valid resolutions. This is consistent with the factorization-gap analysis in Eq. (4): the individual actions remain plausible, but independent sampling recombines them into invalid joint outcomes.

At the ambiguous state, the corridor reduces to an anti-coordination game with two optimal joint actions, a structure known to be difficult for independently acting agents in cooperative multi-agent learning (Claus & Boutilier, 1998; Matignon et al., 2012). DMM instead produces a strongly correlated joint-action distribution by allowing agents' evolving intents to influence subsequent votes before commitment. Although DMM is trained with $K = 4$, evaluating the same checkpoints with 8 and 12 refinement rounds further increases the valid-action frequency (Figure 1); the full test-time-depth and teacher-forcing ablations are reported in Appendix H.2.

Table 2: Corridor joint-action frequencies (mean ± 95% Student-t CI; 1,000 draws per seed). Valid: WL=(wait, left), RW=(right, wait); invalid: WW=(wait, wait), RL=(right, left).

	Valid		Invalid	
Method	WL (%)	RW (%)	WW (%)	RL (%)
DMM ($\beta = 0.8$)	51.4 $\pm$9.0	43.4 $\pm$9.4	2.2 $\pm$0.5	3.0 $\pm$1.0
DMM ($\beta = 0$)	24.5 $\pm$0.4	25.6 $\pm$0.5	23.3 $\pm$0.2	26.6 $\pm$0.2
LC-MAPF	25.7 $\pm$0.1	24.2 $\pm$0.2	22.7 $\pm$0.4	27.4 $\pm$0.2
MAGAT+	24.6 $\pm$0.3	24.9 $\pm$0.4	23.5 $\pm$0.4	26.9 $\pm$0.2
HMAGAT	25.6 $\pm$1.4	24.7 $\pm$2.0	22.5 $\pm$2.2	27.1 $\pm$1.4

With a zero teacher-forcing floor, DMM loses its concentration on the two valid resolutions and approaches the near-uniform frequencies of LC-MAPF, MAGAT+, and HMAGAT. Thus, a nonzero teacher-forcing floor is important for learning the correlated refinement behavior.

6 CONCLUSION

In this work, we identified a limitation of decentralized MAPF policies that sample agents' final actions independently: individually plausible choices can still combine into an incompatible joint action. DMM addresses this by refining stochastic action intents through local communication before simultaneous commitment, allowing agents' evolving choices to influence one another while preserving decentralized execution. We further introduced MICPO, a critic-free reinforcement-learning method that fine-tunes this multi-round decision process from shared task-level outcomes.

Across POGEMA, DMM generally achieves higher success rates and lower solution costs than the evaluated learned policies, while MICPO further improves both DMM variants over their imitation-pretrained counterparts. At the solver level, DMM-MICPO-3M solves 1,598 of 1,600 MovingAI tasks, achieving the highest coverage among all evaluated methods. The compact DMM-MICPO-0.8M further demonstrates scalability, solving every tested large-scale instance up to 1,048,576 simultaneously acting agents.

A remaining limitation is that DMM does not enforce joint-action feasibility as a hard constraint. Although MICPO penalizes blocked action proposals through the task-level objective, incompatible intermediate choices are not explicitly excluded during refinement. Collision-free execution may therefore still require an external mechanism such as CS–PIBT, which makes the system centralized in our MovingAI and large-scale experiments. Incorporating explicit feasibility constraints into the refinement process is a natural direction for future work.

Overall, these results show that coupling agents' evolving action choices before commitment can improve decentralized MAPF beyond independent one-shot action sampling.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the writing of proofs: the authors stated the decentralized factorization gap and wrote its proof, and AI tools were then used to check that argument for errors and to polish and simplify its presentation. We have not used generative AI tools to generate synthetic datasets, to develop theoretical models or conceptual frameworks, to formulate mathematical claims, to provide critical ingredients for proving mathematical claims, to propose or refine hypotheses, to design or provide feedback on our research methodology or experiments, or to interpret results; translation, dataset cleaning and reformatting, and qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for editing the software code, for drafting and editing parts of this paper to improve its readability and formatting, and for identifying and summarizing relevant literature. We have reviewed all AI-assisted work. The research idea, the DMM refinement mechanism, the MICPO algorithm, and the architecture of each component were designed and specified by the authors; all AI-assisted code was reviewed and tested by the authors, and every experimental number reported in this paper was produced by running the code; the proof was verified by the authors, who are responsible for its correctness; all AI-assisted text was read and revised by the authors; and all references were added manually by the authors, who checked each one against its original source. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The DMM intent-refinement mechanism, the imitation-pretraining objective, and the MICPO fine-tuning procedure are specified in Section 4. The DMM-3M and DMM-0.8M architectures are detailed in Appendix C, and the complete MICPO objective, matched-group construction, reward definitions, trajectory filtering, and multi-round replay procedure, together with pseudocode, are given in Appendix D.3. All pretraining and fine-tuning hyperparameters are reported in Appendix D.4, and the training data, teacher-forcing schedules, POGEMA domain configurations, large-scale protocol, and corridor protocol in Appendix E.

For the evaluation protocols, the MovingAI benchmark composition, solver budgets, reported quantities, and the Repeat-State Escape procedure are described in Appendix F; the million-agent scenarios, timing, and memory measurements in Section 5 and Appendix G; and the GPU-accelerated environment, observation pipeline, and chunking strategies in Appendix G.2. The assumptions and the complete proof of the decentralized factorization gap are given in Appendix B.3.

An anonymized code repository, containing the DMM implementation, the checkpoints used for evaluation, the training and evaluation configurations, and the raw per-instance benchmark logs needed to recompute the reported comparisons, is available at https://anonymous.4open.science/r/dmm-1DB6/README.md; the checkpoints are additionally included in the anonymized supplementary archive accompanying this submission, which also contains a video demonstration of DMM's behavior.

REFERENCES

Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. Advancing learnable multi-agent pathfinding solvers with active fine-tuning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10564–10571, 2025a.

Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. MAPF-GPT: Imitation learning for multi-agent pathfinding at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23126–23134, 2025b.

Tomoki Arita and Keisuke Okumura. Local guidance for configuration-based multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29296–29304, 2026.

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4): 819–840, 2002.

Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, Oded Betzalel, David Tolpin, and Eyal Shimony. Icbs: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, pp. 223–225, 2015.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 746–752, 1998.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46375–46408, 2025.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Wei Fu, Chao Yu, Zelai Xu, Jiaqi Yang, and Yi Wu. Revisiting some common practices in cooperative multi-agent reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 6863–6877, 2022.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018.

Chengyang He, Tianze Yang, Tanishq Duhan, Yutong Wang, and Guillaume Sartoretti. Alpha: Attention-based long-horizon pathfinding in highly-structured areas. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14576–14582, 2024.

Chengyang He, Tanishq Duhan, Gadiel Sznaier Camps, Fangyuan Wang, Yuhong Cao, Jiankai Sun, Ge Sun, Mac Schwager, and Guillaume Sartoretti. PRIMAL3: Pathfinding via reinforcement and imitation multi-agent learning-leveraging LaCAM3. *arXiv preprint arXiv:2608.04905*, 2026.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the learning of non-autoregressive transformers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 9356–9376, 2022.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 4651–4664, 2021.

Rishabh Jain, Keisuke Okumura, Michael Amir, Pietro Lio, and Amanda Prorok. Pairwise is not enough: Hypergraph neural networks for multi-agent pathfinding. In *International Conference on Learning Representations*, 2026a.

Rishabh Jain, Keisuke Okumura, Michael Amir, and Amanda Prorok. Graph attention-guided search for dense multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29504–29512, 2026b.

He Jiang, Yutong Wang, Rishi Veerapaneni, Tanishq Harish Duhan, Guillaume Adrien Sartoretti, and Jiaoyang Li. Deploying ten thousand robots: Scalable imitation learning for lifelong multi-agent path finding. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

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

Jeric Lew, Yuhong Cao, Derek Ming Siang Tan, and Guillaume Sartoretti. Aid: Agent intent from diffusion for multi-agent informative path planning. In *International Conference on Swarm Intelligence*, pp. 41–54, 2026.

Jiaoyang Li, Daniel Harabor, Peter J Stuckey, Ariel Felner, Hang Ma, and Sven Koenig. Disjoint splitting for multi-agent path finding with conflict-based search. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pp. 279–283, 2019.

Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. MAPF-LNS2: Fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 10256–10265, 2022.

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

Zhiyuan Li, Wenshuai Zhao, Lijun Wu, and Joni Pajarinen. AgentMixer: Multi-agent correlated policy factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18611–18619, 2025.

Zhuoran Li, Hai Zhong, Xun Wang, Qingxin Xia, Lihua Zhang, and Longbo Huang. Diffusing to coordinate: Efficient online multi-agent diffusion policies. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Jinhao Liang, Jacob K. Christopher, Sven Koenig, and Ferdinando Fioretto. Simultaneous multi-robot motion planning with projected diffusion models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 37162–37180, 2025.

Jinhao Liang, Sven Koenig, and Ferdinando Fioretto. Discrete-guided diffusion for scalable and safe multi-robot motion planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 23417–23424, 2026a.

Jinhao Liang, Sven Koenig, and Ferdinando Fioretto. Simulation-informed diffusion for decentralized multi-robot motion planning. *arXiv preprint arXiv:2605.27697*, 2026b.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. GDPO: Group reward-decoupled normalization policy optimization for multi-reward RL optimization. *arXiv preprint arXiv:2601.05242*, 2026a.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. LLM collaboration with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32150–32158, 2026b.

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

Ziyuan Ma, Yudong Luo, and Hang Ma. Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA 2021)*, pp. 8699–8705, 2021.

Ziyuan Ma, Yudong Luo, and Jia Pan. Learning selective communication for multi-agent path finding. *IEEE Robotics and Automation Letters*, 7(2):1455–1462, 2022.

Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: A survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11655–11662, 2023.

Keisuke Okumura. Engineering LaCAM*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 1501–1509, 2024.

Keisuke Okumura, Manao Machida, Xavier Défago, and Yasumasa Tamura. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310:103752, 2022.

Thomy Phan, Joseph Driscoll, Justin Romberg, and Sven Koenig. Confidence-based curriculum learning for multi-agent path finding. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*, pp. 1558–1566, 2024.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yorai Shaoul, Itamar Mishani, Shivam Vats, Jiaoyang Li, and Maxim Likhachev. Multi-robot motion planning with diffusion models. In *International Conference on Learning Representations*, 2025.

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

Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. POGEMA: A benchmark platform for cooperative multi-agent pathfinding. In *International Conference on Learning Representations*, 2025.

Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pp. 151–158, 2019.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pp. 1261–1263, 2010.

Pavel Surynek, Ariel Felner, Roni Stern, and Eli Boyarski. Efficient SAT approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, pp. 810–818, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rishi Veerapaneni, Arthur Jakobsson, Kevin Ren, Samuel Kim, Jiaoyang Li, and Maxim Likhachev. Work smarter not harder: Simple imitation learning with CS-PIBT outperforms large-scale imitation learning for MAPF. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10229–10236, 2025.

Valeriy Vyaltsev, Alsu Sagirova, Anton Andreychuk, Oleg Bulichev, Yuri Kuratov, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. Learning to communicate locally for large-scale multi-agent pathfinding. *arXiv preprint arXiv:2605.07637*, 2026.

Jiangxing Wang, Deheng Ye, and Zongqing Lu. More centralized training, still decentralized execution: Multi-agent conditional policy factorization. In *International Conference on Learning Representations*, 2023a.

Yuanzhe Wang, Tian Zhi, Zihang Wei, Hongguang Wang, Jiaming Guo, Yang Zhao, Zisheng Liu, Shiyu Quan, Xing Hu, Zidong Du, et al. Discrete diffusion for complex and congested multi-agent path finding with sparse social attention. *arXiv preprint arXiv:2605.13296*, 2026.

Yutong Wang, Bairan Xiang, Shinan Huang, and Guillaume Sartoretti. Scrimp: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9301–9308, 2023b.

Satosi Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development*, 4(1):66–82, 1960.

Zhengbang Zhu, Minghuan Liu, Liyuan Mao, Bingyi Kang, Minkai Xu, Yong Yu, Stefano Ermon, and Weinan Zhang. Madiff: Offline multi-agent learning with diffusion models. *Advances in Neural Information Processing Systems*, 37:4177–4206, 2024.

Appendix Contents

Section	Contents
Appendix A	Extended discussion of classical, learned, and refinement-based MAPF methods
Appendix B	Formal MAPF and communication setting, factorization-gap proof and example
Appendix C	DMM-3M and DMM-0.8M architecture variants and implementation details
Appendix D	Training data and objectives, MICPO procedure, and hyperparameters
Appendix E	POGEMA protocol, refinement-depth and intent-communication ablations
Appendix F	MovingAI protocol, Repeat-State Escape, and decoding comparison
Appendix G	Million-agent scenarios, measurements, and GPU implementation
Appendix H	Corridor protocol, refinement-depth and teacher-forcing ablations

A Extended Related Work

Existing MAPF approaches can be broadly divided into classical methods, which rely on predefined planning or coordination procedures, and learning-based methods, which acquire their decision rules from data. We first review classical approaches spanning explicit search, optimization, and reactive coordination, then turn to learning-based methods and their approaches to decentralized coordination.

A.1 Classical MAPF Approaches

Conflict-Based Search (CBS) (Sharon et al., 2015) and its improved variants (Boyarski et al., 2015; Li et al., 2019) are canonical search-based MAPF methods. They systematically explore the joint state space and can provide optimal or bounded-suboptimal guarantees, but are often limited in scalability.

Reduction-based approaches reformulate MAPF as an equivalent well-studied optimization problem, such as minimum-cost flow or Boolean satisfiability (SAT), and use existing solvers to compute optimal or near-optimal solutions (Surynek et al., 2016; Lam et al., 2022).

Fast rule-based solvers such as PIBT (Okumura et al., 2022) instead rely on simple local coordination rules to achieve high scalability, though they generally sacrifice optimality. Building on PIBT, LaCAM (Okumura, 2023; 2024) integrates PIBT as a low-level policy within a search-based framework, combining reactive local decisions with conflict-aware global reasoning to improve solution quality. Similarly, MAPF-LNS2 (Li et al., 2022) employs large neighborhood search with adaptive repair strategies for rapid generation and refinement of near-optimal solutions. Simpler approaches like prioritized planning (Ma et al., 2019) trade optimality for runtime efficiency and remain popular in large-scale MAPF scenarios due to their computational simplicity.

These classical approaches make different trade-offs between solution quality, computational cost, and scalability. Search-based methods can provide stronger guarantees but typically incur increasing computational cost as the number of agents grows, while reactive and prioritized methods achieve greater scalability through more local decision-making.

A.2 Learning-Based MAPF Approaches

Learning-based approaches learn coordination policies from data, offering an alternative to the predefined planning and coordination procedures used by classical MAPF methods. They differ in whether an agent commits to its action in one shot. Methods with *direct commitment* select an action from a predicted distribution in one step, with any resulting conflicts handled separately or left unresolved; methods with *refinement before commitment* revise an intermediate action choice over several passes before acting.

Direct Commitment. One of the pioneering works, PRIMAL (Sartoretti et al., 2019), showed that decentralized agents using a learned policy could solve MAPF when the only shared information is

the agents' targets, without any further inter-agent communication. Also without inter-agent communication, MAPF-GPT (Andreychuk et al., 2025b) instead uses a Transformer-based architecture trained via imitation learning on a large dataset of expert MAPF solutions. MAPF-GPT-DDG (Andreychuk et al., 2025a) further fine-tunes MAPF-GPT on additional expert data collected via active learning. SILLM (Jiang et al., 2025) scales imitation learning to lifelong MAPF with 10,000 agents. Concurrently with this work, PRIMAL3 (He et al., 2026) also targets scale, reporting single-instance stress tests with up to 100,000 agents at 20% obstacle density.

To enable richer coordination, DHC (Ma et al., 2021) brought learned communication (Sukhbaatar et al., 2016; Foerster et al., 2016) to MAPF, with agents exchanging latent representations, improving performance over PRIMAL. DCC (Ma et al., 2022) refines this with selective communication, deciding when and what to communicate to limit redundancy while preserving necessary information.

A related line of work uses graph attention to structure communication. MAGAT (Li et al., 2021) replaces a fixed communication structure with learned attention over neighboring agents, letting each agent weight incoming messages by relevance, and is trained via imitation learning on expert demonstrations, similar to MAPF-GPT. MAGAT+ (Jain et al., 2026b) extends this with three stacked attention layers in place of the single layer used in the original, and adopts a two-stage imitation-learning paradigm: it is first pretrained on trajectories from a search-based expert, then fine-tuned with additional imitation data on the target map. HMAGAT (Jain et al., 2026a) instead replaces the pairwise graph with a hypergraph representation to capture group-level interactions among several neighboring agents at once, likewise trained purely by imitation learning.

A different family of methods builds communication into a Transformer rather than a graph attention mechanism. SCRIMP (Wang et al., 2023b) keeps a separate convolutional observation encoder and fuses neighboring agents' messages through a dedicated Transformer-based communication block. LC-MAPF (Vyaltsev et al., 2026) instead uses a Transformer encoder-decoder architecture for the whole pipeline, exchanging local latent representations over multiple communication rounds before each agent samples a single final action.

Across these approaches, communication enriches what each agent's policy conditions on, but the final action is sampled once, independently, from each agent's distribution after communication ends: even a multimodal policy can then recombine locally valid choices into an infeasible joint action.

Cooperative multi-agent reinforcement learning also recognizes that independent per-agent policies cannot express coordinated joint behavior, but existing remedies either relax decentralization or give up correlation at execution: Fu et al. (2022) order agents autoregressively and broadcast each action to successors, AgentMixer (Li et al., 2025) correlates policies only in training, and MACPF (Wang et al., 2023a) recovers an equally valued factorizable policy with a single mode. DMM does neither.

Refinement Before Commitment. Iterative refinement is the mechanism generative models such as diffusion use to represent multimodal distributions, and a recent line brings it to multi-agent planning. In discrete MAPF, DiffLNS (Wang et al., 2026) is a concurrent example: it centrally refines the joint action tensor of all agents into a full-horizon plan and passes the result to LNS2 for repair, motivated, like this work, by the multimodality of the expert distribution. In contrast, DMM performs decentralized, per-timestep intent refinement through communication and commits the resulting actions directly.

A larger body of related work considers continuous-space multi-robot motion planning, where refinement is likewise centralized in most cases (Liang et al., 2025; Shaoul et al., 2025; Liang et al., 2026a), although decentralized variants exist. In these variants, coordination is introduced separately from refinement: by inferring or simulating teammates while refining alone (Zhu et al., 2024; Liang et al., 2026b), by a critic that couples agents only during training (Li et al., 2026), or by exchanging already planned trajectories (Lew et al., 2026). Thus, communication typically provides either context for refinement or a decision already formed. DMM instead communicates the intermediate refinement state itself, allowing agents to condition on one another's intents while they are still forming.

B Formal Setting and Factorization Gap

B.1 MAPF Preliminaries

A MAPF instance is a tuple $\big(G, \{s_u\}_{u\in U}, \{g_u\}_{u\in U}\big)$, where $G = (V, E)$ is a four-connected grid, $U = \{u_1, \ldots, u_n\}$ is the set of agents, and $s_u, g_u \in V$ are the start and goal vertices of agent u, respectively. Starts are pairwise distinct, as are goals. Time is discrete. The (joint) configuration at timestep t is $\mathbf{v}^t = (v_u^t)_{u\in U} \in V^n$, with $\mathbf{v}^0 = (s_u)_{u\in U}$. At each timestep, each agent selects an action from the common action set $\mathcal{A} = \{\text{wait}, \text{up}, \text{down}, \text{left}, \text{right}\}$, forming the joint action $\mathbf{a}^t = (a_u^t)_{u\in U} \in \mathcal{A}^n$.

A joint action is *feasible* at $\mathbf{v}^t$ if every move is either a wait action or traverses an edge in E, no two agents occupy the same vertex after the transition, and no two agents traverse the same edge in opposite directions during the same timestep. We denote the set of feasible joint actions at configuration $\mathbf{v}$ by $\mathcal{F}(\mathbf{v}) \subseteq \mathcal{A}^n$.

A solution is a sequence of feasible joint actions that reaches $v_u^T = g_u$ for all $u \in U$ for some $T \leq H$, where H is the execution horizon.

We evaluate solution quality using the *sum of costs* (SoC) and *makespan* (MS). Let c_u denote the earliest timestep at which agent u reaches its goal and remains there for the remainder of the episode. The SoC measures the total arrival cost across all agents, whereas the makespan measures the arrival time of the last agent:

$$\text{SoC} = \sum_{u\in U} c_u, \qquad \text{MS} = \max_{u\in U} c_u. \tag{9}$$

We additionally report the *success rate* (SR), defined as the fraction of instances for which all agents reach their goals within H timesteps.

B.2 Decentralized MAPF with communication

We model decentralized MAPF as a finite-horizon Dec-POMDP (Bernstein et al., 2002), defined by the tuple $M = \langle S, \mathcal{A}, U, P, R, O, \mathcal{O}\rangle$, where a state $s^t \in S$ comprises the agent configuration $\mathbf{v}^t$ and the fixed targets. Given the current state and an executed joint action, the transition function P determines the next state. The reward function $R : S \times \mathcal{A}^n \to \mathbb{R}$ assigns a single scalar reward shared by all agents. Rewards are terminal and undiscounted.

At timestep t, agent u receives the local observation $o_u^t = \mathcal{O}_u(s^t) \in O$, consisting of an egocentric $(2\rho + 1) \times (2\rho + 1)$ patch centered at v_u^t. The Dec-POMDP is augmented with a communication channel represented by a dynamic directed graph $G_{\text{comm}}^t = (U, E_c^t)$. Each agent u communicates with its k nearest agents within its observation range, including itself, denoted $\mathcal{N}^t(u)$.

Each timestep consists of K synchronous communication rounds followed by a single action commitment. For $r = 1, \ldots, K$, let

$$x_u^{t,r} = \big(o_u^t,\, \{m_w^{t,r'}\}_{w\in\mathcal{N}^t(u),\, r'<r}\big) \tag{10}$$

denote the information available to agent u at the start of communication round r, comprising its local observation o_u^t and the messages $m_w^{t,r'}$ received from its neighbors in preceding rounds. In particular, $x_u^{t,1} = (o_u^t, \emptyset)$. Agent u then emits

$$m_u^{t,r} \sim \mu_\theta\big(\cdot \mid x_u^{t,r}\big), \tag{11}$$

where μ_θ is the message-generation distribution shared by all agents; it may be deterministic, as in conventional learned communication, or depend on randomness private to agent u, as in DMM. Messages are generated simultaneously in each round and subsequently transmitted along G_{comm}^t. After the K rounds, let $x_u^t = x_u^{t,K+1}$ denote the information available to agent u at action commitment. All agents share the parameters of a stochastic policy $\pi_\theta(a_u^t \mid x_u^t)$ and select their actions simultaneously.

Decentralized execution is required to satisfy:

(i) Agent u conditions its decision only on x_u^t.

(ii) No agent observes another agent's committed action before selecting its own, and no agent ordering is imposed.

(iii) No component encodes the global state s^t, predicts the joint action $\mathbf{a}^t$, or directly aggregates global information; communication is restricted to local neighbors.

Under (i)–(iii), per-agent computation is independent of the number of agents n, and a decentralized solution reduces to a collection of local policies; communication augments x_u^t but leaves the transition function P and reward R unchanged.

B.3 Decentralized Factorization Gap Proof

We use the notation introduced in the factorization-gap analysis in the main text; all expectations, entropies, and mutual informations are computed under $P^\star$. For any product executor Q_q, the expected joint KL divergence decomposes as

$$\begin{aligned}
&\mathbb{E}_X D_{\mathrm{KL}}\left(P^\star(A \mid X) \,\middle\|\, \prod_{u\in U} q_u(A_u \mid X_u)\right) \\
&\quad = \underbrace{\mathrm{TC}(A \mid X)}_{\text{action coupling}} + \underbrace{\sum_{u\in U} I(A_u; X_{-u} \mid X_u)}_{\text{missing local information}} \\
&\qquad + \underbrace{\sum_{u\in U} \mathbb{E}_{X_u} D_{\mathrm{KL}}(P^\star(A_u \mid X_u) \,\|\, q_u(A_u \mid X_u))}_{\text{learning error}}.
\end{aligned} \tag{12}$$

To prove this identity, expand the product form of Q_q:

$$\begin{aligned}
&\mathbb{E}_X D_{\mathrm{KL}}\left(P^\star(A \mid X) \,\middle\|\, \prod_{u\in U} q_u(A_u \mid X_u)\right) \\
&\quad = -H(A \mid X) + \sum_{u\in U} \mathbb{E}[-\log q_u(A_u \mid X_u)].
\end{aligned} \tag{13}$$

Each summand is a local cross-entropy and therefore satisfies

$$\begin{aligned}
\mathbb{E}[-\log q_u(A_u \mid X_u)] = H(A_u \mid X_u) \\
+ \mathbb{E}_{X_u} D_{\mathrm{KL}}(P^\star(A_u \mid X_u) \,\|\, q_u(A_u \mid X_u)).
\end{aligned} \tag{14}$$

Next, because $X = (X_u, X_{-u})$,

$$H(A_u \mid X_u) = H(A_u \mid X) + I(A_u; X_{-u} \mid X_u). \tag{15}$$

Substituting this identity into the expanded KL and collecting entropy terms gives Eq. (12), since

$$\sum_{u\in U} H(A_u \mid X) - H(A \mid X) = \mathrm{TC}(A \mid X). \tag{16}$$

All three terms in Eq. (12) are nonnegative. With sufficient capacity and ideal cross-entropy optimization, each local policy converges to its Bayes-optimal conditional,

$$q_u^\star(a_u \mid x_u) = P^\star(A_u = a_u \mid X_u = x_u), \tag{17}$$

and the learning-error terms vanish. The remaining irreducible error is therefore

$$\mathrm{TC}(A \mid X) + \sum_{u\in U} I(A_u; X_{-u} \mid X_u). \tag{18}$$

This is the minimum in Eq. (4).

Interpretation and corridor example

For a fixed context x, the KL term defining $\mathrm{TC}(A \mid X = x)$ is zero exactly when the expert joint distribution factorizes into its per-agent marginals. Positive total correlation is sufficient, but not necessary, for a strict decentralized gap: the missing-information term can remain positive when $\mathrm{TC}(A \mid X) = 0$. If every factor instead observes the same full context, $X_u = X$ for all u, that term vanishes.

The resulting restriction is analogous to the multimodality problem of non-autoregressive sequence models: independently learned token marginals can mix parts of different valid translations (Gu et al., 2018). The full-context case reduces to the conditional-total-correlation bound of Huang et al. (2022); decentralized execution additionally incurs the penalty for information unavailable in each X_u. Related separations between independent and correlated policies concern value decomposition (Fu et al., 2022) or attainable return (Wang et al., 2023a). Our gap instead concerns matching the expert joint distribution targeted by per-agent imitation.

The symmetric corridor makes the correlation term concrete. With g for "go" and w for "wait," suppose that at its ambiguous context x the expert chooses the two coordinated resolutions equally often:

$$P^\star((g, w) \mid x) = P^\star((w, g) \mid x) = \tfrac{1}{2}. \tag{19}$$

Each exact local marginal is uniform over $\{g, w\}$, so independent sampling assigns probability $1/4$ to every pair, including the collision (g, g) and the stall (w, w). Consequently,

$$\mathrm{TC}(A \mid X = x) = 2\left(\frac{1}{2}\log\frac{1/2}{1/4}\right) = \log 2 > 0. \tag{20}$$

The marginals are correct; their product is not. Deterministic communication can reduce missing local information by moving some of X_{-u} into X_u, but once its transcript is fixed, independent final samples still form a product distribution. Exchanging agents' stochastic choices before commitment is one way to escape that restriction while retaining local, simultaneous action selection.

C DMM Architecture Variants

Section 4 defines the DMM intent-refinement mechanism at the policy level. We instantiate this mechanism with two network architectures, DMM-3M and DMM-0.8M, shown in Figure 6. Both use the same action-intent representation, intent-augmented messages, local communication graph, and iterative voting procedure. They differ in how the local observation is encoded and how the resulting observation representation is combined with the changing messages at each refinement round.

DMM-3M. The DMM-3M architecture is shown in Figure 6(a). DMM-3M retains the observation encoder and communication-decoder structure of LC-MAPF (Vyaltsev et al., 2026). Each agent receives the 256-token observation described in Appendix D.2. Tokens are embedded at width 192 and processed by three Transformer (Vaswani et al., 2017) encoder blocks. A set of 32 learned latent queries then cross-attends to the encoded observation (Jaegle et al., 2021), producing a fixed 32×96 observation representation for the current environment step.

During each refinement round, the intent-augmented messages of up to 13 local agents are gathered and concatenated with the 32 observation latents. The resulting sequence, containing at most 45 tokens of width 96, is processed by three Transformer decoder blocks. A learned action/message query then cross-attends to this sequence to produce the per-round feature used by the policy and message heads. These heads output the five action logits ϕ_u^r and the next message feature h_u^r, respectively. The observation latents remain unchanged across refinement rounds; only the intent and communication state evolve. DMM-3M contains 3,241,784 trainable parameters.

DMM-0.8M. The DMM-0.8M architecture is shown in Figure 6(b). DMM-0.8M preserves the same DMM refinement interface but replaces the Transformer observation backbone and full per-round decoder with a structured, lighter-weight architecture. It uses the same 256-token observation layout as DMM-3M, whose two informative components are the 11×11 spatial field and 13 local agent records of 10 tokens each.

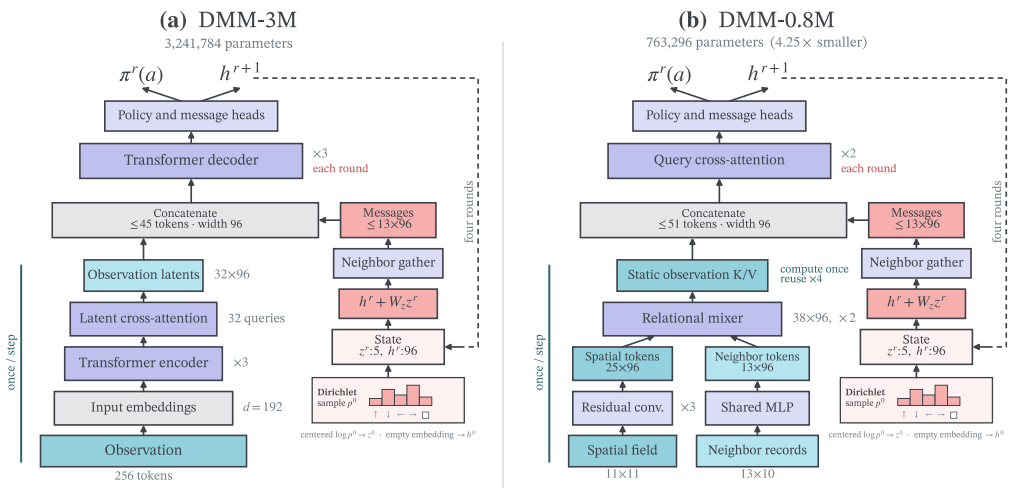

Figure 6: DMM architecture variants. **(a)** DMM-3M retains the Transformer encoder–decoder backbone of LC-MAPF. The tokenized observation is compressed into 32 latent tokens, which are combined with the current neighbor messages and processed by the decoder at every refinement round. **(b)** DMM-0.8M uses a structured observation encoder that separately processes the spatial field and local agent records. The resulting observation tokens are mixed once per environment step, and their cross-attention key/value projections are reused across refinement rounds; only the current messages and query state change between rounds. Both architectures implement the same DMM intent-refinement process.

The 121 spatial tokens are first embedded at width 96 and arranged as an 11×11 feature map. A convolutional stem followed by three residual convolutional blocks processes this map, after which adaptive pooling produces a 5×5 representation, or 25 spatial tokens. In parallel, each 10-token agent record is embedded and processed by a shared MLP, producing 13 agent tokens of width 96. Learned spatial-position, agent-slot, and token-type embeddings distinguish the two sources. The 25 spatial and 13 agent tokens are then concatenated and processed by two relational self-attention blocks, yielding a 38-token observation representation.

This representation is computed once per environment step. To reduce the computation repeated during intent refinement, DMM-0.8M uses two query cross-attention blocks in place of the DMM-3M decoder. For each block, the key and value projections of the 38 observation tokens are precomputed once and reused across all refinement rounds. At round r, the current intent-augmented messages are gathered from up to 13 local agents and projected dynamically. A learned query attends jointly to the static observation representation and these round-specific messages, and the updated query is passed through the two cross-attention blocks sequentially. The final 96-dimensional query feature is mapped by the same type of policy and message heads to the action logits ϕ_u^r and next message feature h_u^r.

DMM-0.8M therefore moves most observation-dependent computation outside the refinement loop while retaining fresh message-dependent computation at every round. It contains 763,296 trainable parameters, making it about $4.25\times$ smaller than DMM-3M while leaving the intent update, communication semantics, and final action rule unchanged.

D Training and MICPO

D.1 Training Objectives

Let

$$\mathcal{D}_E = \left\{ \left(\{o_{u,j}\}_{u \in U_j}, \{a^\star_{u,j}\}_{u \in U_j}, G_{\mathrm{comm},j} \right) \right\}_{j=1}^{N_E} \tag{21}$$

denote an expert demonstration dataset of N_E timestep samples, where U_j is the set of agents in sample j, $o_{u,j}$ is agent u's local observation, $a^\star_{u,j}$ is its expert action, and $G_{\mathrm{comm},j}$ specifies the

communication graph. For a given sample, the communication process in Appendix B induces the context $x_{u,j}$ available to agent u at action commitment.

For conventional action-based imitation, the shared policy is trained by minimizing the per-agent cross-entropy

$$\mathcal{L}_{\mathrm{CE}}(\theta) = -\frac{1}{N_E}\sum_{j=1}^{N_E}\frac{1}{|U_j|}\sum_{u\in U_j}\log \pi_\theta\left(a^\star_{u,j} \mid x_{u,j}\right). \tag{22}$$

This objective matches each agent's action distribution to the corresponding expert action conditioned on the information available at commitment.

Under independent final-action sampling, each agent samples its action from the shared policy conditioned on its local context after the K communication rounds. For a fixed timestep, suppressing the sample and time indices for clarity, the resulting joint-action distribution factorizes as

$$Q_{\pi_\theta}(\mathbf{a} \mid x) = \prod_{u\in U}\pi_\theta(a_u \mid x_u), \qquad x = (x_1, \ldots, x_n). \tag{23}$$

Thus, communication may affect each agent's action distribution through its local context x_u, but under independent final-action sampling the action choices remain conditionally independent given x.

A reinforcement-learning objective instead optimizes the expected team return over decentralized rollouts. Let

$$\zeta = (s^0, \mathbf{a}^0, s^1, \mathbf{a}^1, \ldots, s^T) \tag{24}$$

denote a rollout, where $T \le H$ is the termination timestep, reached when the joint goal configuration is achieved or the horizon is exhausted. Given the terminal team return $R(\zeta) \in \mathbb{R}$, the objective is

$$J(\theta) = \mathbb{E}_{\zeta\sim\pi_\theta}\left[R(\zeta)\right]. \tag{25}$$

D.2 DMM Observations and Training

Each agent receives a tokenized observation of up to 256 tokens encoding its 11×11 local field of view, agent-specific attributes, and spatial context. It communicates with up to 13 agents within a 5-cell radius, including itself. The imitation-pretraining dataset is derived from LC-MAPF (Vyaltsev et al., 2026) and aggregates samples from *mazes*, *random*, and *house* maps with up to 32 agents in a 0.6:0.2:0.2 split; the house-map generator follows (He et al., 2024). Each sample contains tokenized observations, expert actions, and local communication adjacency information.

Both DMM variants undergo 1,000,000 imitation-pretraining iterations before MICPO fine-tuning. Initial-intent and round-level teacher-forcing probabilities start at 1.0 and are annealed over the first 100,000 iterations to the retained floor $\beta_0 = \beta_r = 0.8$. DMM-3M uses AdamW with cosine learning-rate decay and requires approximately 272.8 GPU-hours across four H100 GPUs. Its MICPO fine-tuning uses 500 outer iterations (96,000 optimizer updates) with group size $G = 24$ and requires 56.3 GPU-hours across four H100 GPUs. DMM-0.8M has an effective imitation batch size of 800 and requires approximately 100.3 GPU-hours across four H100 GPUs for pretraining. Its MICPO fine-tuning also uses 500 outer iterations (96,000 optimizer updates) with $G = 24$, requiring approximately 14.1 GPU-hours across four H100 GPUs. Further hyperparameters are in Table 3.

D.3 MICPO Training Procedure

Section 4 summarizes MICPO. We give its exact objective, matched-group construction, and implementation details here (Algorithm 1).

MICPO objective and matched groups

MICPO optimizes a shared team objective without a value function. A centralized critic must generalize over a combinatorially large joint state, whereas a decentralized critic has only partial information about the global outcome. MICPO instead compares rollouts under matched conditions, adapting group-relative optimization (Shao et al., 2024) to DMM's multi-agent, multi-round setting. Group-relative optimization has also been applied to multi-agent LLM collaboration (Liu et al.,

2026b) and refined with step-level groups of recurring anchor states (Feng et al., 2025); here, groups share the sampled initial intent at every step, and importance ratios are computed per agent and refinement round.

Matched rollout groups. At each optimization iteration, we sample B scenarios and construct M matched groups of G trajectories per scenario. At every environment step t, the G trajectories within a group share the same sampled initial intent z_t^0, while subsequent refinement votes are sampled independently. Thus, trajectories are matched on the scenario and on z_t^0 at each step, but their states may diverge due to earlier stochastic refinement decisions.

Team return and group-relative advantage. For rollout ζ, let H_ζ denote its number of environment transitions and define the off-goal duration of agent u as

$$T_{\mathrm{off},u}(\zeta) = \sum_{t=0}^{H_\zeta - 1} \mathbf{1}\left[v_u^t \neq g_u\right]. \tag{26}$$

Let $B_u(\zeta)$ denote the number of blocked non-wait actions proposed by agent u during the rollout. We form two trajectory-level components:

$$q_c(\zeta) = -\frac{1}{|U|} \sum_{u \in U} \log(1 + T_{\mathrm{off},u}(\zeta)), \tag{27}$$

$$q_b(\zeta) = -\frac{1}{|U|} \sum_{u \in U} B_u(\zeta). \tag{28}$$

Their weighted sum defines the unnormalized team return:

$$R(\zeta) = w_c q_c(\zeta) + w_b q_b(\zeta). \tag{29}$$

Because the components have different scales, we normalize them separately within each matched group of G trajectories, as in GDPO (Liu et al., 2026a). We define

$$N_G(x) = \begin{cases} \dfrac{x - \mu_G(x)}{\sigma_G(x)}, & \sigma_G(x) \geq \tau, \\ 0, & \sigma_G(x) < \tau, \end{cases} \tag{30}$$

where $\mu_G(x)$ and $\sigma_G(x)$ are the mean and standard deviation within the matched group, and $\tau > 0$ prevents division by a near-zero quantity. The group-relative advantage is

$$\hat{A}(\zeta) = N_G(w_c N_G(q_c(\zeta)) + w_b N_G(q_b(\zeta))). \tag{31}$$

This advantage is shared across all agents and communication rounds of the corresponding rollout.

Bounded replay. Replaying every collected decision is expensive because episodes can be long and each environment step contains K refinement rounds. Within each matched group, we rank trajectories by $R(\zeta)$ and retain the κ highest- and κ lowest-return trajectories. Advantages are computed from all G trajectories before filtering, so replay selection does not alter the comparison group used for credit assignment. From each retained trajectory, we sample S environment timesteps for optimization.

Round-level policy optimization. MICPO applies policy optimization to individual refinement votes. Recall that $p_{\theta,u}^r = \mathrm{softmax}(\phi_u^r)$ is the round-r action distribution of agent u, and define $p_{\mathrm{old},u}^r$ and $p_{\mathrm{ref},u}^r$ analogously for the old and frozen reference policies. The importance ratio for the sampled vote y_u^r is

$$\rho_u^r = \frac{p_{\theta,u}^r(y_u^r)}{p_{\mathrm{old},u}^r(y_u^r)}. \tag{32}$$

For each selected environment timestep, all K refinement rounds are replayed together. The stored initial intent and vote sequence reconstruct the intent trajectory, while message features are recomputed across rounds so gradients propagate through the complete communication pathway.

The clipped objective is averaged over agents and rounds:

$$\mathcal{L}_{\text{clip}} = -\mathbb{E}_{u,r}\left[\min\left(\rho_u^r \hat{A},\ \text{clip}\left(\rho_u^r, 1-\varepsilon, 1+\varepsilon\right)\hat{A}\right)\right]. \tag{33}$$

To limit drift from coordinated behavior learned during imitation pretraining, we regularize toward a frozen reference policy initialized from the imitation-pretrained checkpoint, similar to (Shao et al., 2024). Its penalty is evaluated at each agent and communication round:

$$\mathcal{L}_{\text{KL}} = \alpha_{\text{KL}} \mathbb{E}_u\left[\sum_{r=1}^{K} D_{\text{KL}}\left(p_{\theta,u}^r \,\|\, p_{\text{ref},u}^r\right)\right]. \tag{34}$$

The final objective is $\mathcal{L} = \mathcal{L}_{\text{clip}} + \mathcal{L}_{\text{KL}}$. The reference anchors each round's action distribution while the communication pathway remains trainable.

Training scenario generation. Training scenarios are generated online. For each scenario, the map geometry and start–goal configuration are generated from independent random seeds. Training uses a mixture of maze and randomly obstructed maps, with maze sampling probability p_{maze}. Randomly obstructed maps use obstacle density p_{obs}; maps with disconnected free-space regions are rejected and resampled before agent starts and goals are generated.

The number of agents N is fixed within a training run. Map height H_{map} and width W_{map} are sampled independently from the ranges $[H_{\text{map}}^{\min}, H_{\text{map}}^{\max}]$ and $[W_{\text{map}}^{\min}, W_{\text{map}}^{\max}]$, respectively. Varying the map dimensions exposes the policy to different agent densities and congestion levels while keeping N fixed. Start and goal locations are sampled on the generated map using POGEMA.

Rollout collection. At each environment timestep, DMM executes all K refinement rounds before committing an environment action. A trajectory ends when all agents simultaneously occupy their goals or when the maximum rollout horizon $T_{\max}$ is reached. Agents are not absorbed at their goals and may move away again on subsequent timesteps.

For policy replay, the implementation retains the observation, communication neighborhood, sampled initial intent z_t^0, the sequence of refinement votes $y_t^{1:K}$, and the corresponding log-probabilities under π_{old} for each environment timestep.

Trajectory filtering and timestep sampling. For each matched group, $R(\zeta)$ and $\hat{A}(\zeta)$ are first computed for all G collected trajectories as defined in Section 4. The trajectories are then ranked by $R(\zeta)$, and the κ trajectories with the lowest returns together with the κ trajectories with the highest returns are retained. The retained trajectories use the advantages computed from all G trajectories.

For each retained trajectory ζ, S valid environment timesteps are sampled uniformly. Sampling is performed without replacement when the trajectory contains at least S valid timesteps and with replacement otherwise. Timestep sampling is applied only along the environment-time dimension: selecting timestep t retains all K refinement rounds associated with that decision.

Multi-round replay. Intermediate message features are not stored in the replay data. For a selected timestep, replay begins from the stored observation, neighborhood information, and initial intent z_t^0. The stored vote sequence $y_t^{1:K}$ determines the successive intent updates, while the policy is evaluated sequentially through all K rounds to recompute the corresponding message features and action distributions.

The stored old-policy log-probabilities are used in the round-level importance ratios defined in Section 4. The same replay sequence is evaluated under the frozen reference policy to compute the categorical KL terms. At the beginning of each MICPO iteration, π_{old} is synchronized with the current policy π_θ, whereas π_{ref} remains fixed at the imitation-pretrained checkpoint.

D.4 Training Hyperparameters

Table 3 summarizes the imitation-pretraining and MICPO configurations used for DMM-3M and DMM-0.8M. Architectural details are given in Appendix C, and the MICPO rollout and replay procedure is described in Appendix D.3.

Algorithm 1 MICPO procedure

Initialize π_θ from imitation pretraining
Set $\pi_{\text{ref}} \leftarrow \pi_\theta$ and keep it frozen
for each MICPO iteration **do**
  Set $\pi_{\text{old}} \leftarrow \pi_\theta$
  Sample B training scenarios
  for each sampled scenario **do**
    for $m = 1, \ldots, M$ **do**
      Initialize a rollout group of G trajectories $\zeta_1, \ldots, \zeta_G$ from the same scenario
      for each rollout timestep t **do**
        Sample an initial intent state z_t^0 and share it across the G trajectories
        Sample refinement votes independently across trajectories for K rounds under π_{old}
        Store the sampled votes and their log-probabilities under the corresponding p_{old}
      end for
    end for
  end for
  for each rollout group **do**
    Compute $R(\zeta_g)$ and $\hat{A}(\zeta_g)$ for $g = 1, \ldots, G$
    Let $\mathcal{I}$ contain the indices of the κ lowest- and κ highest-return trajectories
    for $g \in \mathcal{I}$ **do**
      Sample S valid timesteps from ζ_g
    end for
  end for
  for each replay minibatch **do**
    Replay all K refinement rounds from the stored initial intent z_t^0 and vote sequence $y_{g,t}^{1:K}$
    Recompute message features $h_{u,g,t}^r$ and action distributions $p_{\theta,u,g,t}^r$ and $p_{\text{ref},u,g,t}^r$ for all agents u and rounds $r = 1, \ldots, K$
    Update θ by minimizing $\mathcal{L} = \mathcal{L}_{\text{clip}} + \mathcal{L}_{\text{KL}}$
  end for
end for

E POGEMA Evaluation and Mechanism Ablations

E.1 POGEMA Benchmark Protocol

We follow the LC-MAPF evaluation protocol (Vyaltsev et al., 2026) and compare only learned policies: HMAGAT (Jain et al., 2026a), MAGAT+ (Jain et al., 2026b), MAPF-GPT-85M (Andreychuk et al., 2025b), MAPF-GPT-DDG-2M (Andreychuk et al., 2025a), and LC-MAPF-3M (Vyaltsev et al., 2026). Random and Mazes use grids from 17×17 to 21×21 with up to 96 and 80 agents, respectively; their map families appear in pretraining. Warehouse uses 33×46 maps with up to 192 agents, and Cities-Tiles uses 64×64 maps with up to 256 agents. The latter two differ in topology and agent count from the training data and assess out-of-distribution generalization. Episodes are limited to 128 steps except for Cities-Tiles, which allows 256.

E.2 Inference-Time Refinement Depth

DMM is trained with $K_{\text{train}} = 4$ refinement rounds. To test whether the learned refinement process is tied to this depth, we keep the policy parameters fixed and vary only the number of rounds executed at inference, $K_{\text{test}} \in \{1, 2, 3, 4, 8, 12\}$. We evaluate both DMM-3M before MICPO fine-tuning and DMM-MICPO-3M after fine-tuning, using the same evaluation protocol as in the main POGEMA benchmark. Figure 7 shows success rate, makespan, and agent-agent collisions at the largest team size per domain: 96 agents on Random, 80 on Mazes, 192 on Warehouse, and 256 on Cities-Tiles.

Across all four domains and both training stages, most of the benefit from additional refinement is obtained by the training depth $K_{\text{train}} = 4$. A single refinement round is insufficient in the maximum-agent settings: increasing K_{test} to $K_{\text{test}} = 4$ produces large gains in success rate together with substantial reductions in makespan and agent-agent collisions. Beyond four rounds, the additional

Table 3: Model and training hyperparameters for DMM variants.

Parameter	DMM-3M	DMM-0.8M
Imitation pretraining		
Effective batch size	512	800
Training iterations	1,000,000	1,000,000
Maximum learning rate	6×10^{-4}	6×10^{-4}
Minimum learning rate	6×10^{-5}	6×10^{-5}
Warm-up iterations	2,000	2,000
Intent-update step size (δ)	0.25	0.25
Log-smoothing constant (η)	1×10^{-8}	1×10^{-8}
Group-normalization threshold (τ)	1×10^{-6}	1×10^{-6}
Initial-intent TF start / floor (β_0)	1.0 / 0.8	1.0 / 0.8
Round TF start / floor (β_r)	1.0 / 0.8	1.0 / 0.8
MICPO fine-tuning		
Outer iterations	500	500
Training agents (N)	32	32
Map height / width range	9–13	9–13
Maze probability (p_{maze})	0.5	0.5
Random-map obstacle density (p_{obs})	0.2	0.2
Maximum rollout horizon ($T_{\max}$)	128	128
Scenarios per iteration (B)	4	4
Groups per scenario (M)	3	3
Trajectories per group (G)	24	24
Trajectories retained per extreme (κ)	4	4
Timesteps per retained trajectory (S)	16	16
Clip parameter (ε)	0.2	0.2
KL coefficient (α_{KL})	0.01	0.01
Off-goal weight (w_c)	1.0	1.0
Blocked-action weight (w_b)	0.3	0.3
Learning rate	1×10^{-6}	1×10^{-6}

improvements are smaller. Thus, four refinement rounds capture most of the gains observed up to $K_{\text{test}} = 12$, while additional rounds provide smaller improvements.

Although DMM is trained only with $K_{\text{train}} = 4$, the same checkpoints remain effective when unrolled for additional refinement rounds. At $K_{\text{test}} = 8$ and 12, performance generally matches or improves on the $K_{\text{test}} = 4$ result, indicating that the learned update is not specialized to an exact four-round horizon. Changes beyond the training depth are smaller and occasionally non-monotonic across metrics, but additional rounds often further reduce collisions. Thus, the learned refinement dynamics can be extended at test time without retraining.

MICPO shifts the refinement-depth trade-off toward stronger performance at smaller K_{test}. At the same test-time depth, DMM-MICPO-3M generally attains higher success rates and lower makespan and collision counts than DMM-3M. On Warehouse at 192 agents, for example, DMM-MICPO-3M with $K_{\text{test}} = 2$ already achieves a lower makespan than DMM-3M with $K_{\text{test}} = 12$. Additional rounds remain useful after fine-tuning, but the MICPO checkpoint requires fewer refinement rounds to reach a comparable level of performance.

E.3 Intent-State Communication Ablation

To determine which component of the recurrent message supports coordination, we apply inference-time perturbations to fixed DMM-3M and DMM-MICPO-3M checkpoints without retraining, all evaluated at the trained refinement depth $K = 4$. The agent's own intent update and final action rule remain unchanged; only the information communicated to neighboring agents is modified. "Full" uses the standard message containing both the learned feature h and the projected current intent z; "no-h" communicates only z, and "no-z" communicates only h. In "z^0-only", each agent updates its own intent normally, but broadcasts the fixed initial intent z^0 at every round instead of the evolving z. Finally, "shuffled" permutes the assembled messages across agents before decoding, preserving

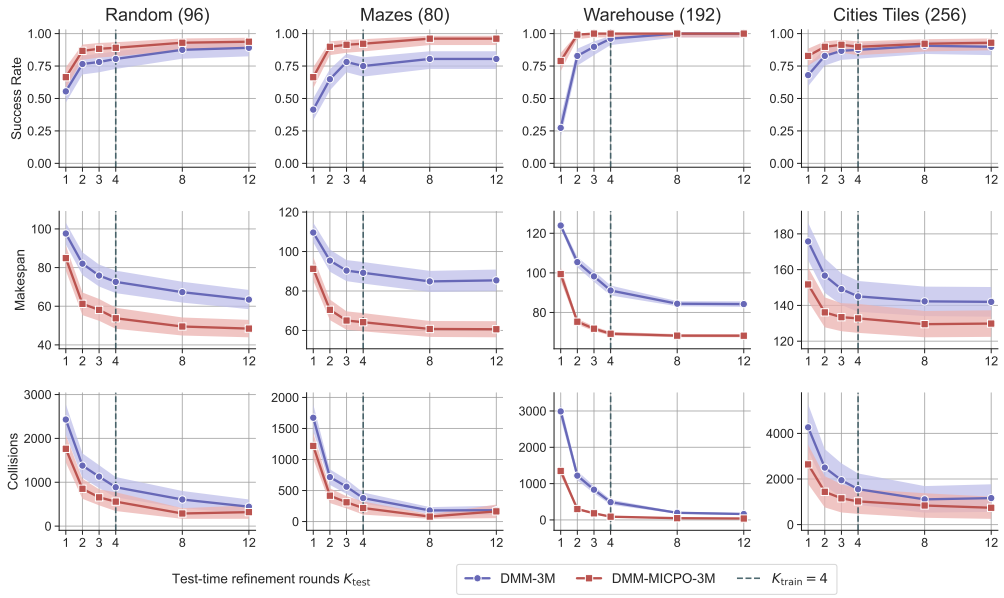

Figure 7: Effect of inference-time refinement depth at the largest evaluated team size in each POGEMA domain. DMM-3M and DMM-MICPO-3M are both trained with $K_{\text{train}} = 4$ refinement rounds; at evaluation, the same fixed checkpoints are executed with $K_{\text{test}} \in \{1, 2, 3, 4, 8, 12\}$ without retraining. Rows report success rate, makespan, and agent-agent collisions, respectively. Shaded regions show 95% confidence intervals across $n = 128$ validation instances per domain, with the same instances used across refinement depths and policies. Wilson score intervals are used for success rate, while makespan and collision counts use normal intervals of the form $\bar{x} \pm 1.96\, s/\sqrt{n}$. The vertical dashed line marks the training depth $K_{\text{train}} = 4$.

the message structure while mismatching content with its originating agent. Figure 8 reports success rate on Mazes across team sizes.

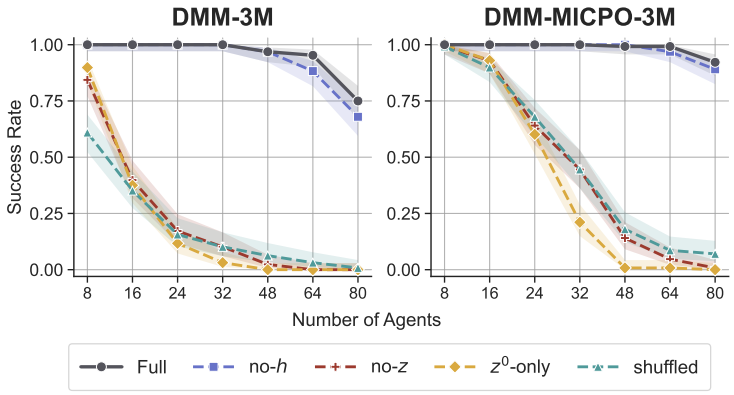

Figure 8: Success rate on Mazes under inference-time perturbations of the communicated state. Both DMM-3M and DMM-MICPO-3M are evaluated with fixed weights at $K = 4$. Full communicates both h and the evolving intent z; no-h removes h from the broadcast, no-z removes z, and z^0-only replaces the evolving communicated intent with its initial value while leaving the agent's own intent dynamics unchanged. Shuffled permutes the assembled messages across agents before decoding. Shaded regions show 95% Wilson score confidence intervals across $n = 128$ validation instances, with the same instances used for all compared communication variants.

The clearest separation is between variants that communicate the evolving intent and those that do not: removing h while retaining z causes only a limited drop relative to "Full", whereas "no-z"

deteriorates sharply as team size grows. For these trained policies, the evolving intent therefore carries more of the coordination-relevant information than h alone.

The "z^0-only" intervention shows this dependence is not explained simply by providing an intent-valued message: each agent updates its own z normally, but neighbors repeatedly receive only the initial z^0, and performance then degrades to a level similar to "no-z". The relevant information thus lies in the evolution of z across refinement rounds, not just its initialization. Together with Appendix E.2, this supports repeated refinement benefiting from agents communicating updated action intents across rounds.

The "shuffled" intervention provides a complementary test: the full messages are preserved, but the assembled message tensors are permuted across agents, breaking the correspondence between an agent and its constructed message set. The resulting drop in success shows retaining message content alone is insufficient once this correspondence is disrupted, so effective use of the evolving intent also depends on communicating it in the correct local context.

MICPO improves robustness to all these perturbations but does not change their qualitative ordering: "Full" and "no-h" remain stronger at large team sizes after fine-tuning, while "no-z", "z^0-only", and "shuffled" still deteriorate sharply. The dependence on evolving intent communication thus persists after policy optimization, even as MICPO improves overall performance.

These are post-training interventions on fixed checkpoints, so they measure which communication components the learned policies rely on, not the performance of architectures retrained without h, without z, or with a different message parameterization.

F MovingAI Evaluation and Decoding

Benchmark composition. Each of the 33 MovingAI maps provides 25 even and 25 random scenarios. We select the maximum-agent instance from every scenario, then omit the 50 tasks from `maze-128-128-1`, for which preliminary runs found that no evaluated method solved any maximum-agent task. This leaves 32 maps and 1,600 tasks with 32–7,070 agents. An unsolved task counts as a failure and cannot contribute a solution-quality win.

Search-based and hybrid solvers. LG-LaCAM and MAPF-LNS2 represent two complementary search paradigms, while LaGAT combines a learned MAGAT+ policy with LaCAM's search structure. Because these methods can continue constructing and exploring alternatives, each run is limited to 600 wall-clock seconds. We request the first valid solution from LG-LaCAM and disable its optional iterative post-solution improvement. The reported runtime is time to the returned solution, or the full budget for an unsolved task.

Reactive learned solvers. HMAGAT, DMM-MICPO-0.8M, and DMM-MICPO-3M act directly in the environment and are limited to 5,000 environment steps. Their action proposals are processed by the same CS–PIBT collision shield. In the main comparison, all three also use Repeat-State Escape (RSE) to prevent a reactive rollout from cycling among previously visited joint configurations. Results without RSE are reported in Appendix F.1.

RSE stores the complete configuration history in a hash set. When CS–PIBT produces a duplicate configuration, RSE considers agents in PIBT priority order and selects the first unfinished agent whose proposed transition can be forbidden without removing all available actions. The prohibition is local to the current timestep, and CS–PIBT is invoked again from the same state to construct a different successor. Constraints accumulate only during this retry process and are discarded after an action is executed. RSE therefore maintains no search tree, open list, rollback operation, or persistent search node; it transfers LaCAM's repeated-state avoidance principle to a purely reactive solver.

Reported quantities. We report the number of solved tasks, per-task virtual-best SoC wins, the distribution of SoC relative to the virtual best on tasks solved by each method, and wall-clock runtime. The virtual best is the minimum SoC obtained by any of the evaluated methods on a task, with ties counted as wins for every tied method. Since the search-based and hybrid solvers are time-

limited whereas the reactive policies are step-limited, runtime reflects the solver configuration rather than an equalized compute budget.

F.1 Decoding and Repeat-State Escape

Table 4 compares three reactive-policy configurations on the same 1,600 MovingAI task keys: sampling without RSE, sampling with RSE, and argmax with RSE. All use PIBT-based shielding. Every mean sum of costs (SoC) and mean makespan uses the same 1,526 tasks solved by all nine configurations, matched by task key; solved counts use all 1,600 tasks. For HMAGAT, sampling and argmax refer to action selection. For DMM, they refer to the four communication rounds; the final action is greedy in all three configurations.

Table 4: MovingAI decoding and RSE configurations. Solved is out of 1,600 tasks. MS: makespan.

Method	Decoding / RSE	Solved ↑	SoC ↓	MS ↓
HMAGAT	Sampling	1,556	362,741	506.3
	Sampling + RSE	1,576	361,858	481.6
	Argmax + RSE	1,543	334,302	534.3
DMM-MICPO-0.8M	Sampling	1,593	369,955	492.6
	Sampling + RSE	1,592	369,956	494.4
	Argmax + RSE	1,593	321,085	460.0
DMM-MICPO-3M	Sampling	1,599	335,999	468.6
	Sampling + RSE	1,599	336,001	468.3
	Argmax + RSE	1,598	311,768	452.8

Adding RSE to sampling raises HMAGAT coverage by 20 tasks and lowers its common-cohort mean makespan from 506.3 to 481.6. For DMM, the same contrast has little effect on coverage or quality: DMM-MICPO-0.8M changes from 1,593 to 1,592 solved tasks and DMM-MICPO-3M stays at 1,599, while their matched mean SoCs change by less than 0.2%. In contrast, the historical argmax-with-RSE configurations have much lower matched SoC than sampling with RSE: 321,085 versus 369,956 for DMM-MICPO-0.8M and 311,768 versus 336,001 for DMM-MICPO-3M. HMAGAT argmax with RSE also lowers matched SoC but solves 33 fewer tasks and has a higher matched makespan than HMAGAT sampling with RSE.

G Million-Agent Scalability and GPU Implementation

G.1 Large-Scale Scenarios and Measurement Protocol

At each population size, DMM-MICPO-0.8M and GPU-PIBT are evaluated on four matched 2304×2304 maze layouts with approximately 29.1% obstacles and an $H = 32768$ horizon. Agent density ranges from 3.48% to 27.86% of traversable cells. Starts and goals are sampled uniformly from traversable cells, with all start and goal cells distinct. Episodes end when all agents reach their goals or the horizon is reached. Unlike the earlier million-agent evaluation of Andreychuk et al. (2025a), this benchmark uses mazes rather than an empty 2048×2048 grid with start–goal distances capped at 64.

DMM proposes a wait without network inference for an agent if it and all agents in its local field of view are at their goals; this condition is reevaluated every step. CS–PIBT still processes every proposed action, including those waits. Environment transitions and observations run on the GPU, with batched BFS and cached distances for cost-to-go observations (Appendix G.2). The GPU CS–PIBT implementation groups agents whose candidate moves may conflict and processes independent groups in parallel. Within each group, it follows standard PIBT priority ordering and backtracking and produces the same joint action as sequential PIBT. At high density, a large conflict group can limit parallelism and make shielding a major runtime cost.

GPU-PIBT instead uses BFS-based move preferences. These are computed across GPUs and collected on one GPU for PIBT conflict resolution; selected actions are then broadcast. For both methods, we report means across the four scenarios and minimum and maximum episode lengths. Mean

step and PIBT times exclude the one-time BFS prefill; PIBT time includes action broadcast. Total runtime includes prefill and subsequent computation and communication. Amortized decision time divides total runtime by agent-steps, including agents for whom network inference was skipped.

G.2 GPU-Accelerated Environment and Observation Pipeline

The original POGEMA implementation generated observations and executed environment transitions on the CPU. While efficient for moderate-scale runs, this setup became a throughput bottleneck when scaling to many agents or running large batches of parallel rollouts, as each step required transferring data between CPU and GPU and limited overall simulation speed.

To overcome this, we reimplement both the environment dynamics and the observation pipeline to run entirely on the GPU. The full environment state (agent positions, goals, obstacle grid, and collision-resolution buffers) is kept in VRAM. At each timestep, a pipeline of CUDA kernels processes all agent actions: candidate moves are proposed, swap conflicts cancelled, and vertex collisions resolved through an iterative cascade using atomic priority arbitration, following the same soft-collision semantics as the original environment. No environment or observation tensors are moved back to the host during the rollout loop.

The observation construction is also rewritten for the GPU. Egocentric cost-to-go maps are obtained from a batched breadth-first search where each agent is handled by a single GPU block using shared memory; the search expands from the agent's goal outward and stops once the local observation window centered on the agent's current position is filled, avoiding full-map traversal. Rather than recomputing distances from scratch at every step, we cache larger raw distance windows. Agents whose Chebyshev distance from the cache center exceeds a margin recompute the BFS; all others extract their observation patch via a dedicated kernel that crops and normalizes the cached distances in one pass, deriving next-action tokens without a new BFS. Neighbor information (relative positions, goal offsets, and action histories) is assembled on-device by a batched CUDA kernel: for each agent we identify visible neighbors within a fixed Chebyshev radius, rank them by Manhattan distance, and encode the features. All observation tokens are concatenated and padded to a fixed context length, yielding a batched tensor in VRAM that feeds directly into the policy network, never leaving the device. The same local neighborhood also defines the sparse communication graph (top-k neighbor indices) used by the policy.

To handle very large agent counts without exhausting GPU memory, we employ two complementary chunking strategies. BFS chunking processes cache-miss agents (or all agents when caching is disabled) in smaller sub-batches, limiting peak allocations for frontier and visited grids. Agent chunking partitions agents inside the policy encoder and, within each communication round, across decoder receivers, bounding the size of activations after a synchronization point that assembles the full per-agent message table. Communication itself is already sparse: each agent attends only to this precomputed top-k set rather than to all other agents; chunking does not alter this graph, nor the computation. Both strategies are semantically equivalent to the monolithic path.

H Corridor Study

H.1 Corridor Protocol

The single-width corridor has one side cell where an agent can yield. Two agents start at opposite ends and must swap goals (Figure 1). Two expert trajectories solve the task, one for each yielding agent. They share exactly one state at which both resolutions remain possible. Of the four joint-action combinations at that state, two are coordinated resolutions, mutual waiting is a stall, and simultaneous movement is a collision.

DMM, LC-MAPF, MAGAT+, and HMAGAT are trained by imitation on the same two expert trajectories with their native observation representations. DMM uses $K = 4$ refinement rounds, LC-MAPF uses four communication rounds, and MAGAT+ and HMAGAT retain their original architectures. All models train for 5,000 iterations. For DMM, both teacher-forcing probabilities start at 1.0 and anneal over the first 1,000 iterations to $\beta_0 = \beta_r = 0.8$. A zero-floor variant anneals both probabilities to zero over the same interval, with all other settings unchanged. For each of five training seeds, we draw 1,000 joint actions at the shared ambiguous state from the trained policy.

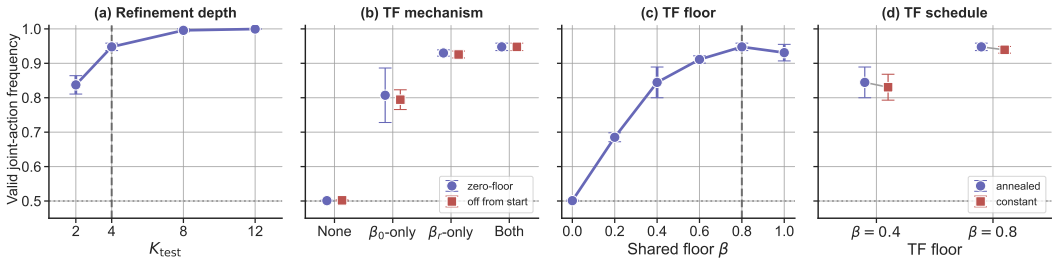

Figure 9: Corridor ablations of refinement depth and teacher forcing. All panels report the valid joint-action frequency $p_{\text{valid}} = p(\text{WL}) + p(\text{RW})$. (a) Checkpoints trained with $K_{\text{train}} = 4$, evaluated at different test-time refinement depths without retraining; the vertical dashed line marks the training depth. (b) Contribution of initial-intent teacher forcing β_0 and round-level teacher forcing β_r: for the blue series, excluded mechanisms are annealed to a zero floor; for the red series, they are disabled from the start of training. (c) Effect of the shared teacher-forcing floor $\beta_0 = \beta_r = \beta$; the vertical dashed line marks the retained floor used in the main DMM training, $\beta = 0.8$. (d) Corridor annealed schedule versus a constant teacher-forcing probability at matched floors. The horizontal dotted line marks the 0.5 valid-action frequency from independent recombination of the expert marginals.

H.2 Refinement and Teacher-Forcing Ablations

We use the ambiguous corridor state to isolate how refinement depth and teacher forcing affect the learned joint-action distribution (Figure 9), reporting the valid joint-action frequency $p_{\text{valid}} = p(\text{WL}) + p(\text{RW})$, where WL and RW are the two coordinated resolutions. Each configuration is evaluated with 1,000 joint-action samples per training seed; point values are means across five seeds, with error bars denoting 95% Student-t confidence intervals across seeds. The horizontal reference at $p_{\text{valid}} = 0.5$ is the independent-sampling outcome for the balanced expert marginals.

Refinement depth. With the training depth fixed at $K_{\text{train}} = 4$, increasing only the test-time refinement rounds raises the valid joint-action frequency from $83.7 \pm 2.7\%$ at $K_{\text{test}} = 2$ to $94.8 \pm 1.1\%$ at $K_{\text{test}} = 4$, and further to $99.6 \pm 0.4\%$ at $K_{\text{test}} = 8$ and $99.9 \pm 0.1\%$ at $K_{\text{test}} = 12$. The same fixed checkpoints thus continue to suppress invalid joint actions when unrolled beyond the training depth. Table 5 shows this increase does not come from collapsing onto a single valid resolution: both WL and RW retain substantial frequency, while the stall and collision frequency drops from 0.16 at $K_{\text{test}} = 2$ to below 0.01 at $K_{\text{test}} = 12$.

Table 5: Effect of test-time refinement depth in the corridor, as percentages. All checkpoints are trained with $K_{\text{train}} = 4$. Entries are mean frequencies across five independently trained checkpoints (one per training seed), with 1,000 joint-action samples per checkpoint. The valid joint-action frequency is $p_{\text{valid}} = p(\text{WL}) + p(\text{RW})$; uncertainty denotes the 95% Student-t confidence interval across checkpoints. WL and RW are the two valid joint actions, while Invalid combines the stall (WW) and collision (RL) outcomes.

		Joint-action frequency (%)		
K_{test}	p_{valid} (%)	WL	RW	Invalid
2	83.7 $\pm$2.7	41.7	42.1	16.3 $\pm$2.7
4 (train)	94.8 $\pm$1.1	51.4	43.4	5.2 $\pm$1.1
8	99.6 $\pm$0.4	52.7	46.9	0.4 $\pm$0.4
12	99.9 $\pm$0.1	52.9	47.0	0.1 $\pm$0.1

Teacher-forcing mechanism. Round-level teacher forcing accounts for most of the coordination effect. Annealing the excluded mechanism to zero gives a valid joint-action frequency of 0.930 with only β_r active, 0.807 with only β_0 active, and 0.501 with both mechanisms removed; disabling the excluded mechanisms from the start of training gives the same ordering, with frequencies of 0.926,

0.794, and 0.502. The standard configuration with both mechanisms reaches 0.948. Since β_r determines whether the expert action is incorporated into the evolving intent update at each round, this is consistent with the communicated intent trajectory being the main teacher-forced signal supporting coordinated refinement.

Teacher-forcing strength and schedule. Increasing the retained teacher-forcing floor improves the valid joint-action frequency up to $\beta = 0.8$, where the standard configuration reaches 0.948; at $\beta = 1.0$ the frequency decreases to 0.931. With a shared floor $\beta_0 = \beta_r = \beta$, the frequencies are 0.501, 0.685, 0.845, 0.911, 0.948, and 0.931 for $\beta = 0$, 0.2, 0.4, 0.6, 0.8, and 1.0, respectively. Annealing rather than holding the teacher-forcing probability constant produces only small differences at matched floors: at $\beta = 0.4$, the frequencies are 0.845 and 0.831, while at $\beta = 0.8$ they are 0.948 and 0.939, respectively. Thus, the retained floor has a much larger effect than whether the teacher-forcing probability is annealed or held constant.

Overall, the corridor ablations show that additional test-time refinement continues to improve coordination beyond the training depth, while round-level teacher forcing provides most of the training-time coordination benefit.