ACTION GRAPH POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Coordinating teams of autonomous agents to achieve a common goal is a foundational challenge in multi-agent reinforcement learning (MARL). From automated warehouses where multiple robots must lift heavy pallets to tactical defense systems where fleets must maneuver jointly to surround a target, success in many real-world applications requires multiple agents to synchronize co-dependent actions to achieve the task. In this paper, we propose Action Graph Policies (AGP) for collaborative MARL, where we make candidate actions, rather than agents, the fundamental units of relational reasoning. We construct what we call an "action graph", whose nodes represent available agent-action pairs and we apply multi-head attention over this graph to learn dependencies among candidate action choices. Information exchange across this graph contextualizes each action with respect to the choices available to the entire team, enabling policies to account for for teammates' actions before selecting their own, without explicitly enumerating the joint action space. Theoretically, we analyze settings where typical MARL approaches suffer from either miscoordination or exponential costs from explicitly modeling joint-action dependencies. Empirically, we demonstrate that AGP achieves higher aggregate performance than fourteen strong related MARL methods over nineteen cooperative tasks drawn from three benchmark suites, Level-Based Foraging, the StarCraft Multi-Agent Challenge (SMAC) and SMACv2, and a *Staffing game* of our own. Through qualitative post-training analyses, we also pair visualizations of learned action graphs with task episodes, demonstrating how action dependencies align with coordinated behavior. With ablations assessing the contribution of the action-graph representation, and computational complexity analysis of AGP, our findings demonstrate the value of explicitly modeling action dependencies for learning policies whose success depends on what agents can accomplish together. Anonymized code can be found at: anonymous.4open.science/r/agp-marl.

1 INTRODUCTION

Multi-Agent Reinforcement Learning (MARL) requires multiple agents to make individual decisions that collectively produce effective team behavior (Oroojlooy & Hajinezhad, 2023; Amato, 2024; Zhang et al., 2021; Wang et al., 2022b; Alqudsi & Makaraci, 2025; Keith & La, 2024). In many tasks, the usefulness of an agent's action depends strongly on the actions chosen by its teammates. A robot may need another robot to lift an object from the opposite side; agents may need to occupy complementary locations; or several agents may need to execute different parts of a maneuver together. In such cases, an action which is effective as part of the right joint behavior may accomplish little when executed alone. Effective cooperation therefore requires agents to account not only for their own action choices, but also for how those choices relate to the possible actions of their teammates.

Learning coordinated behavior in this setting has been proposed under several distinct formulations. Value-factorization and centralized-critic methods (Sunehag et al., 2017; Rashid et al., 2020b; Wang et al., 2020a; Yu et al., 2022; Lowe et al., 2017) use joint information during training to guide policies that act on local information. Communication methods (Sukhbaatar et al., 2016; Das et al., 2019; Sun et al., 2024; Hu et al., 2024; Wojtala et al., 2025) let agents exchange observations, latent features or intentions before they act. Graph-based methods (Jiang et al., 2020; Böhmer et al., 2020; Li et al., 2021; Duan et al., 2024; Gupta et al., 2026; Liu et al., 2025) model relations among agents and use them to structure that exchange or the value function. Autoregressive policies (Wen et al., 2022; Mahjoub et al., 2025) take a different route: they fix an order over the agents and let each agent condition on the actions already chosen.

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.

Despite the differences, current approaches largely organize coordination at the level of agents. As illustrated in Figure 1(a), such representations primarily capture which agents should interact. However, action co-dependence is more fine-grained: the nature of the interaction between two agents can vary across their candidate actions. Some action pairs may be complementary, others conflicting, and others largely independent. Representing these dependencies through a single agent-level relation, or through a fixed order over agents, amounts to an action-agnostic compression that can discard task-relevant variation and obscure the action-specific structure that ultimately determines coordinated behavior.

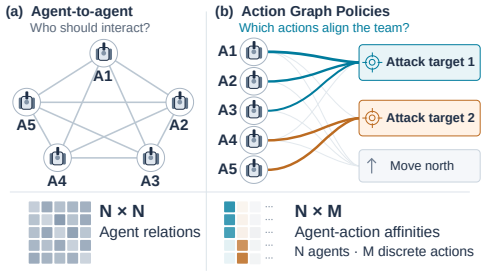


Figure 1: Moving from modeling *which agents interact* to *which actions align*.

In this paper, we therefore shift the unit of coordination from agents to their candidate actions. We construct an *action graph* (Figure 1(b)) with one node per candidate action of each agent. We connect each agent's action nodes to compare its alternatives, and nodes representing the same action across agents to account for teammates choosing that action. Together, these edges evaluate each candidate against the agent's alternatives and in the context of teammates' choices. The graph is sparse, yet any two nodes are at most two hops apart (Section 4), enabling action-level interaction without enumerating the joint action space. We then propose *Action Graph Policies* (AGP) for cooperative MARL, which construct features for each candidate action and use attention-based message passing to learn context-based dependencies across the action graph. Each agent then derives its policy from its own action nodes, accounting for its teammates' possible actions. Our contributions are:

- We introduce a sparse action-graph construction over agent-action pairs that enables explicit reasoning over candidate-action dependencies with efficient connectivity, and we formally characterize its structural and computational properties.
- We develop Action Graph Policies that builds action-specific representations, learns context-dependent dependencies through attention over the action graph, and derives each agent's policy directly from its candidate-action nodes. This enables action-level coordination, without enumerating the joint action space or imposing an autoregressive order.
- We compare AGP with fourteen MARL baselines across nineteen scenarios from three benchmark suites and a controlled staffing game, complemented by scaling studies with up to 500 candidate actions per agent, ablations, and post-hoc analysis of learned action graphs.

2 Preliminaries and Related Work

In this work, we model a task as a cooperative Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein et al., 2002; Oliehoek & Amato, 2016) augmented with communication, given by the tuple $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \mathcal{P}, \mathcal{R}, \{\mathcal{O}_i\}_{i\in\mathcal{N}}, \Omega, \gamma \rangle$. Here, $\mathcal{N} = \{1, \ldots, N\}$ is the agent set, $\mathcal{S}$ the state space, $\mathcal{A}_i$ and $\mathcal{O}_i$ the action and observation spaces of agent i, $\mathcal{P}$ the transition function, $\mathcal{R}$ the shared reward, Ω the observation function, and γ the discount factor. The joint action space is $\mathbf{A} = \prod_{i\in\mathcal{N}} \mathcal{A}_i$. When all agents have the same action space, we write $\mathcal{A} = \mathcal{A}_i$ for all i. At step t, agent i observes $o_{i,t}$, maintains local history $\tau_{i,t}$, and selects $a_{i,t}$. The joint action $\mathbf{a}_t = (a_{1,t}, \ldots, a_{N,t})$ yields reward $\mathcal{R}(s_t, \mathbf{a}_t)$ and transition $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, \mathbf{a}_t)$. The objective is $\max_\pi \mathbb{E}_\pi[\sum_t \gamma^t \mathcal{R}(s_t, \mathbf{a}_t)]$. We consider communication-based execution,

$$\pi(\mathbf{a} \mid \boldsymbol{\tau}) = \prod_{i\in\mathcal{N}} \pi_i(a_i \mid \tau_i, \mathbf{m}_{-i}), \tag{1}$$

where $\mathbf{m}_{-i}$ denotes information communicated by teammates; independent execution is recovered with $\mathbf{m}_{-i} = \varnothing$. Training follows centralized training with decentralized execution (CTDE): a centralized value function, or *critic*, uses the global state to estimate the team return during training, while the executed policies use only local histories and communicated messages. Also see Appendix A.1.

Many MARL methods encourage coordinated actions without directly representing dependencies between specific candidate actions of different agents. Value-factorization methods (Sunehag et al., 2017; Rashid et al., 2020b; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2020a) combine individual utilities into a team value, while centralized-critic methods (Lowe et al., 2017; Foerster

et al., 2018; Yu et al., 2022) use global information during training to shape individual policies. Action-centric methods model the semantics or structure of an agent's own actions (Wang et al., 2020b; 2021; Hu et al., 2021). Across these approaches, dependencies between agents' candidate actions are learned implicitly through the value function, critic, or policy. Communication methods instead exchange learned agent-level information (Sukhbaatar et al., 2016; Das et al., 2019; Guan et al., 2022; Hu et al., 2023; Sun et al., 2024), with graph-based variants learning which agents should interact or how information should be routed (Jiang et al., 2020; Liu et al., 2020; 2023; Hu et al., 2024; Wojtala et al., 2025; Liu et al., 2025). Although such communication can encode rich coordination, its representation is typically organized around agents; relations between their particular candidate actions remain implicit. Coordination graphs explicitly represent action compatibility through payoff factors defined over a graph of agents (Guestrin et al., 2002; Kok & Vlassis, 2006; Böhmer et al., 2020; Li et al., 2021; Wang et al., 2022a; Yang et al., 2022; Kang et al., 2022; Duan et al., 2024; Gupta et al., 2026). A common pairwise form is

$$Q_{\text{tot}}(\boldsymbol{\tau}, \mathbf{a}) = \sum_i f_i(\tau_i, a_i) + \sum_{(i,j)\in E} f_{ij}(\tau_i, \tau_j, a_i, a_j). \tag{2}$$

Here, action interactions are explicit but contained within factors indexed by agent pairs. Extending a factor to d agents requires $|\mathcal{A}|^d$ values for a shared action space, making higher-order interactions increasingly costly; Section 3 studies this issue formally. Autoregressive policies provide another form of explicit coupling by factorizing $\pi(\mathbf{a} \mid \boldsymbol{\tau}) = \prod_i \pi_i(a_i \mid \tau_i, a_{1:i-1})$ (Wen et al., 2022; Mahjoub et al., 2025; Li et al., 2025; Fu et al., 2022), but do so through an ordered dependence among agents' actions. Thus, existing approaches either leave across-agent action relations implicit, embed them within agent-indexed factors, or impose an action ordering.

AGP instead makes agent-action candidates the units of relational reasoning. We construct a sparse action graph over these candidates and learn context-dependent interactions among them, so that dependencies between possible actions are represented directly rather than through an agent-level relation. This provides explicit action-level coordination within a communication-based MARL policy, without enumerating the joint action space or introducing an autoregressive action order. Related insights that actions themselves can form the structure of strategic interactions have also been explored in game theory through Action-Graph Games (Jiang et al., 2011; Bhat & Leyton-Brown, 2012) and in mean-field formulations (Yang et al., 2018). AGP develops this perspective into a learned policy representation for cooperative MARL. Section 3 motivates the representation formally, and Section 4 develops the action graph and AGP architecture. Also see Appendix A for more details.

3 Motivation: Action Co-dependence as a Representation Problem

We now ask a more specific question: *where should a policy represent the dependence between candidate actions of different agents?* If the value of agent i choosing a changes according to whether teammate j chooses b, then the relevant relation is not only between agents i and j, but between the candidates (i, a) and (j, b). We call this property *action co-dependence*. The observations below are intended to motivate this representation choice, building on known coordination and factorization challenges in MARL (Claus & Boutilier, 1998; Castellini et al., 2019; Son et al., 2019; Peysakhovich & Lerer, 2018; Böhmer et al., 2020). For clarity, we state the notion using immediate reward; the same idea extends naturally to action values when consequences are delayed. For $a, a' \in \mathcal{A}_i$, define their relative value under teammate actions $\mathbf{a}_{-i}$ as

$$\Delta_i(a, a'; \mathbf{a}_{-i}, s) = \mathcal{R}(s, (a, \mathbf{a}_{-i})) - \mathcal{R}(s, (a', \mathbf{a}_{-i})). \tag{3}$$

Definition 3.1 (Action co-dependence)**.** At state s, agent i's actions are *co-dependent* on a coalition $C \subseteq \mathcal{N} \setminus \{i\}$ if there exist $a, a' \in \mathcal{A}_i$ and two teammate assignments differing exactly on C between which $\Delta_i(a, a'; \mathbf{a}_{-i}, s)$ changes, while changing the actions of any proper subset of C alone leaves it unchanged. This co-dependence has *order* $|C| + 1$, and we say that (i, a) is co-dependent with the candidates (j, b_j), $j \in C$, of the second assignment.

We use *Level-Based Foraging* (Christianos et al., 2020; Papoudakis et al., 2021) as a running example (see Appendix C). Each agent i and each food item f has a level, ℓ_i and ℓ_f. Let $J_f(s, \mathbf{a})$ denote the agents adjacent to f that choose LOAD under the joint action $\mathbf{a}$. The item is collected only when

$$\sum_{i \in J_f(s,\mathbf{a})} \ell_i \geq \ell_f. \tag{4}$$

Consider tasks where every item's level exceeds every agent's, so no agent can satisfy Eq. 4 alone. The value of one LOAD therefore depends on teammates' LOAD choices: an agent's LOAD is co-dependent on the other loaders the item needs; a co-dependence of order equal to the number of loaders (Definition 3.1). When these relations are not represented explicitly, the policy must discover them through observed joint outcomes. The difficulty is visible under a *product policy*, $\pi(\mathbf{a} \mid \boldsymbol{\tau}) = \prod_i \pi_i(a_i \mid \tau_i)$, that is, Eq. 1 with $\mathbf{m}_{-i} = \varnothing$, under which the agents' actions are independent given their histories. If a co-dependence of order L requires L agents to select prescribed actions, each with probability at most p, a product policy samples the whole pattern with probability at most p^L, which is $|\mathcal{A}|^{-L}$ under uniform exploration over a shared action space (see Appendix B: Proposition B.1). So, if five specific agents must all choose LOAD from six available actions, uniform exploration samples this coordinated pattern with probability $6^{-5} \approx 0.013\%$, or roughly once in 7,776 samples. This observation is independent of the learning algorithm: before coordination has been learned, rewarding action combinations become increasingly unlikely to be sampled as more agents must align. This is consistent with known coordination and relative-overgeneralization difficulties in MARL (Panait et al., 2006; Böhmer et al., 2020). Coordination graphs expose action compatibility more directly, typically through factors such as $f_{ij}(\tau_i, \tau_j, a_i, a_j)$ (Guestrin et al., 2002; Kok & Vlassis, 2006; Böhmer et al., 2020). The interaction is stored inside a factor indexed by an agent pair, which limits the construction in two distinct ways. A sum of pairwise factors cannot represent a co-dependence of order three or more; and raising the factor order in a coordination graph restores the expressivity at a combinatorial price, since a factor over L agents has $|\mathcal{A}|^L$ entries and one for every L-agent coalition needs $\binom{N}{L}|\mathcal{A}|^L$ (see Appendix B, Proposition B.2). With $N = 10$, $|\mathcal{A}| = 6$, and $L = 5$, each factor has $6^5 = 7{,}776$ entries, and covering all $\binom{10}{5} = 252$ coalitions requires about 1.96 million entries. Communication can supply the missing context and substantially improve coordination, but typical approaches carry that context in agent-level messages; how a teammate's message affects each particular candidate action is still learned inside the policy. Autoregressive policies condition later decisions on earlier actions (Wen et al., 2022; Mahjoub et al., 2025). In the example, the first loader in the order commits before any teammate has chosen, so its decision can depend on its successors only through what it has learned to expect of them, and the pattern is learned through the fixed order. The chain-rule factorization can represent any joint distribution, so the limitation is not expressivity: before the pattern is learned, uniform conditionals give the same $|\mathcal{A}|^{-L}$ sampling probability; the order is one of $N!$ modeling choices that the task does not single out; and execution takes one sequential step per agent (Appendix J). These limitations motivate the need for a more explicit modeling of interactions among candidate actions. In the following sections, we construct action graphs for this purpose and show that AGP enables effective coordination while avoiding higher-order joint-action tables or imposing an action order.

4 ACTION GRAPH POLICIES

Motivated by the limitations in Section 3, we propose *Action Graph Policies* (AGP) in this section, which make candidate-action relationships explicit through an action-graph representation.

4.1 ACTION GRAPHS

We define a shared discrete action vocabulary $\mathcal{A} = \{a_1, \ldots, a_M\}$, where each $a \in \mathcal{A}$ denotes the same (or a semantically aligned) action wherever it is available. Homogeneous teams share the full vocabulary, whereas heterogeneous teams may expose only a subset $\mathcal{A}_i(s) \subseteq \mathcal{A}$ to agent i at state s, with unavailable actions masked. For example, LBF agents share the same six actions, whereas a StarCraft unit may share movement actions but differ in additional capabilities. Applications with disjoint native action sets also fit the same construction: either domain knowledge can align related actions to shared types (e.g., by target or task), or a common action (such as NO-OP) can be added.

Definition 4.1 (Action graph)**.** For $\mathcal{N} = \{1, \ldots, N\}$ agents with a shared action vocabulary $\mathcal{A} = \{a_1, \ldots, a_M\}$, we define the *action graph* as the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{N} \times \mathcal{A}$ contains one node (i, a) for every agent $i \in \mathcal{N}$ and action $a \in \mathcal{A}$, and its edges are

$$\begin{aligned} \mathcal{E}_{\text{within}} &= \{((i,a),(i,b)) : i \in \mathcal{N},\ a, b \in \mathcal{A},\ a \neq b\}, \\ \mathcal{E}_{\text{across}} &= \{((i,a),(j,a)) : i, j \in \mathcal{N},\ i \neq j,\ a \in \mathcal{A}\}, \\ \mathcal{E} &= \mathcal{E}_{\text{within}} \cup \mathcal{E}_{\text{across}}. \end{aligned} \tag{5}$$

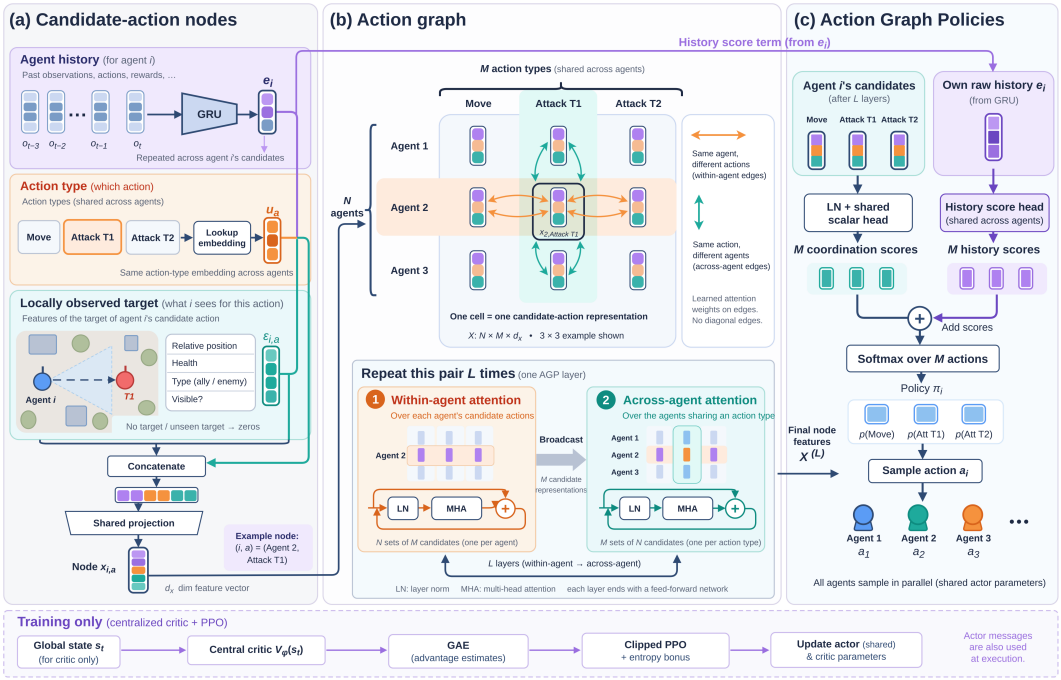


Figure 2: **Overview of Action Graph Policies (AGP).** (a) Each node represents one candidate action of one agent using the agent's local history, the action identity, and target features wherever applicable. (b) Information is exchanged across agents' candidate actions via the Action Graph. (c) Each updated candidate representation is converted into an action score to form the AGP policy.

This forms an $N \times M$ array, with one row per agent and one column per action. Within-agent edges connect nodes along each row, allowing an agent's candidate actions to interact, while across-agent edges connect nodes along each column, exchanging information about the same or semantically aligned actions across agents. Dependencies between different actions of different agents can then propagate through combinations of the two edge types.

Proposition 4.2 (Sparse global connectivity of the action graph)**.** *For $N, M \geq 2$, the full, unmasked action graph has NM nodes and $|\mathcal{E}| = NM(N + M - 2)$ edges, compared with $\mathcal{O}(N^2M^2)$ for a fully connected graph over the same NM nodes. Its topology is the Cartesian product $K_N \square K_M$, where K_N and K_M are complete graphs over the N agents and M actions, respectively, and any two candidate-action nodes are connected by a path of length at most two.*

Proof. See Appendix B. □

Thus, the full action graph is sparse while retaining global multi-hop connectivity. Candidates sharing an agent or action type, such as two LOAD candidates, are directly connected; otherwise, (i, a) and (j, b) are connected through an intermediate node, e.g., $(i, a) \to (i, b) \to (j, b)$. This row-column structure therefore relates arbitrary candidate actions, in particular every co-dependent pair (Definition 3.1), without requiring dense all-to-all connectivity.

4.2 ACTION GRAPH POLICY ARCHITECTURE

AGP first constructs a representation for every candidate action, updates these representations over the action graph, and converts the updated representations into action probabilities (Figure 2).

Candidate-action representations. Agent i encodes its local history τ_i with a GRU (Cho et al., 2014), producing $\mathbf{e}_i \in \mathbb{R}^{d_e}$. Each action a has a learnable embedding $\mathbf{u}_a \in \mathbb{R}^{d_u}$ shared across agents. When the action targets an entity, $\boldsymbol{\epsilon}_{i,a} \in \mathbb{R}^{d_\epsilon}$ contains its locally observed features (Wang et al., 2020b); otherwise it is zero. The initial representation of candidate (i, a) is

$$\mathbf{x}_{i,a}^{(0)} = \mathbf{W}_x[\mathbf{e}_i \parallel \mathbf{u}_a \parallel \boldsymbol{\epsilon}_{i,a}] \in \mathbb{R}^{d_x}, \tag{6}$$

where $\mathbf{W}_x$ is shared across agents and actions. Each node therefore captures the agent's current information, the action being considered, and the entity affected by that action when applicable.

Reasoning over the action graph. We use multi-head attention (MHA) (Vaswani et al., 2017), with pre-layer normalization and residual connections (Xiong et al., 2020), to exchange information along the two edge types. Inspired by axial attention (Ho et al., 2019), each AGP layer processes the two dimensions of the $N \times M$ action array in sequence rather than attending over all NM nodes jointly. Let $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times M \times d_x}$ denote the node representations at layer l. The first update relates the candidate actions of each agent,

$$\hat{\mathbf{x}}_{i,a}^{(l)} = \mathrm{MHA}\left(\mathbf{X}_{i,:}^{(l)}, \mathbf{X}_{i,:}^{(l)}, \mathbf{X}_{i,:}^{(l)}\right)_a, \tag{7}$$

and the second exchanges information across agents for the corresponding action,

$$\mathbf{x}_{i,a}^{(l+1)} = \mathrm{MHA}\left(\hat{\mathbf{X}}_{:,a}^{(l)}, \hat{\mathbf{X}}_{:,a}^{(l)}, \hat{\mathbf{X}}_{:,a}^{(l)}\right)_i. \tag{8}$$

Unavailable actions are masked in both updates. Resulting attention coefficients are the learned edge weights of the dynamic action graph. The number of layers determines how far information can propagate when agents may have co-dependencies over different actions. Let $\mathcal{F}_{i,a}^{(L)}$ denote the candidate actions that can influence (i, a) after L AGP layers.

Proposition 4.3 (Receptive field of AGP)**.** *For an available candidate (i, a), define $\mathcal{F}_{i,a}^{(0)} = \{(i, a)\}$ and, for $l \geq 0$,*

$$\mathcal{F}_{i,a}^{(l+1)} = \bigcup_{j:\, a \in \mathcal{A}_j(s)} \; \bigcup_{b \in \mathcal{A}_j(s)} \mathcal{F}_{j,b}^{(l)}. \tag{9}$$

Then $\mathcal{F}_{i,a}^{(L)}$ contains exactly the candidates that can influence (i, a) through L graph layers. If $\mathcal{A}_i(s) = \mathcal{A}$ for every agent, then $\mathcal{F}_{i,a}^{(L)} = \mathcal{N} \times \mathcal{A}$ for all $L \geq 1$. More generally, one action available to every agent suffices for every available candidate to have a global receptive field for $L \geq 2$.

Proof. See Appendix B. □

This result also clarifies that shared actions provide paths through which information can propagate between otherwise different candidate actions, while additional AGP layers expand this receptive field through chains of overlapping action sets. For disjoint action sets, two agents can communicate through a third agent: two layers suffice if it shares the same action with both, and three if it shares different actions. The updates also have a symmetry. The across-agent update of Eq. 8 attends over the set of matching candidates, so it does not depend on their order, and because every parameter is shared across agents, relabeling the agents permutes the outputs in the same way: the policy is equivariant to agent relabeling whenever agent identity is not part of the observation (see Appendix B.6).

Proposition 4.4 (Computational cost)**.** *One AGP layer in Eqs. 7–8 requires $\mathcal{O}(NM(N + M)d_x)$ attention operations, compared with $\mathcal{O}(N^2M^2d_x)$ for full attention over all NM action nodes.*

Proof. See Appendix B. □

Forming and learning the policy. After L layers, let $\mathbf{h}_{i,a} = \mathbf{x}_{i,a}^{(L)}$. Each candidate is mapped to a coordination-aware score and combined with a standard action score from the agent's history:

$$\pi_i(a_i = a \mid \tau_i, \mathbf{m}_{-i}) = \mathrm{softmax}_a\left[\mathbf{w}^\top \mathrm{LN}(\mathbf{h}_{i,a}) + (\mathbf{W}_b \mathbf{e}_i)_a\right], \tag{10}$$

where the softmax is restricted to available actions and $\mathbf{m}_{-i}$ denotes candidate-action information communicated by teammates. We train the shared actor parameters end to end with PPO (Schulman et al., 2017) under the MAPPO framework (Yu et al., 2022), with the centralized critic only used during training. Communication is part of the differentiable actor, and the action graph's attention weights are learned directly from the task objective without an auxiliary communication loss. At execution, agents form their candidate-action representations from local histories and communicated messages and apply Eq. 10, with no global state or explicit joint-action enumeration required. Each layer communicates Md_x values per agent. To reduce this cost for large action spaces, we also study a low-rank factorization of the $M \times d_x$ representation matrix, using smaller rank K to reduce the message to $K(M + d_x)$ values while preserving its action-specific structure. The factorization is trained end to end with the policy, and we study its empirical trade-offs further in Section 6.

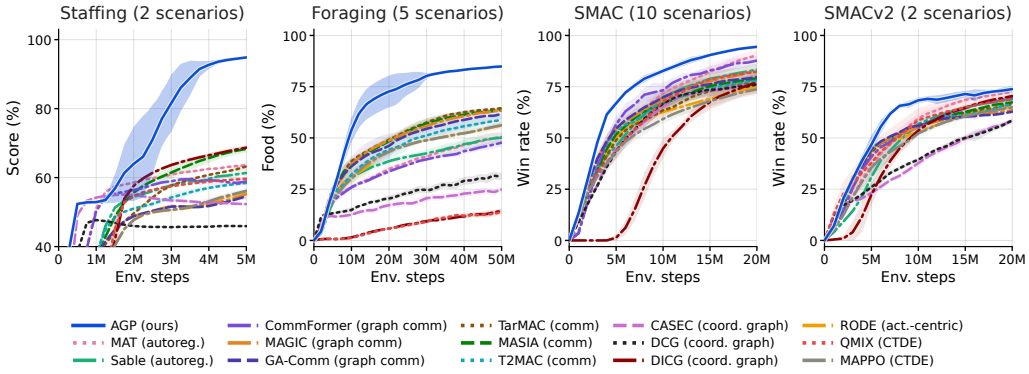


Figure 3: Aggregate learning curves on *Staffing*, LBF, SMAC, and SMACv2. AGP achieves the strongest performance across all suites against fourteen strong MARL methods. Also see Appendix F.

5 Experiments

We evaluate on nineteen scenarios: five *LBF* tasks (Christianos et al., 2020; Papoudakis et al., 2021), ten *SMAC* maps (Samvelyan et al., 2019), two *SMACv2* scenarios (Ellis et al., 2023), and two settings of a controlled *staffing game* introduced in this work (see Appendix C). In *staffing*, at each round, K tasks are active from a pool of T tasks, and N agents each choose a task to join. Each active task k has a value and a required team size r_k, and reward is maximized when the agents coordinate their choices so that each task receives its required number of agents. Since each task is a candidate action, increasing T directly scales the action space while keeping the underlying coordination problem fixed. In our experiments, we use $N = 8$, $K = 4$, and $T \in \{10, 50\}$ in the main comparison and scale to $T \in \{200, 500\}$ to study scalability. We report food collected for LBF, win rate for SMAC/SMACv2, and normalized attainable value for staffing, all in percent.

We compare against fourteen MARL methods (Appendix D): communication (TarMAC, MASIA, T2MAC), graph communication (CommFormer, MAGIC, GA-Comm), coordination graphs (CASEC, DCG, DICG), an action-centric method (RODE), CTDE baselines (MAPPO, QMIX), and autoregressive policies (MAT, Sable). All methods use the same budget within each suite: 50M environment steps on LBF, 20M on SMAC/SMACv2, and 5M on staffing, with five independent trials per experiment. We evaluate 32 episodes every 1M steps on LBF and StarCraft and every 0.25M on staffing. For learning curves, we report the mean across trials with 95% intervals over trials as the shaded region, and suite-level curves average tasks equally within each suite. For final scores, we average evaluation performance over the last tenth of training. Also see Appendix F.

Main Results. Figure 3 shows a consistent pattern across the four suites. The largest gains appear on staffing and LBF, where successful outcomes directly depend on several agents selecting compatible actions. On staffing, AGP reaches 95% of the attainable value, compared with 70% for the strongest baseline, DICG. Here, agents must distribute themselves across tasks without under- or over-staffing them; representing each task choice separately allows AGP to reason directly about which teammates are considering the same assignment. On LBF, AGP collects 84% of the available food compared with 64% for TarMAC. This setting presents the same coordination problem in a different form: a LOAD action is useful only when enough teammates commit to compatible LOAD actions at the same food item. AGP also achieves the strongest aggregate performance on SMAC and SMACv2. The margins are smaller than on staffing and LBF, but the setting is broader: coordination must coexist with movement, targeting, changing observations, and, in SMACv2, episode-to-episode variation in unit configurations. The results therefore show that the benefit of action-level reasoning is not restricted to tasks explicitly constructed around synchronized choices. Across the evaluated suites, AGP attains the strongest aggregate performance among the fifteen methods, while the learning curves show particularly early and persistent separation on the tasks with the clearest action co-dependence. *Scaling the number of candidate actions:* We also probe

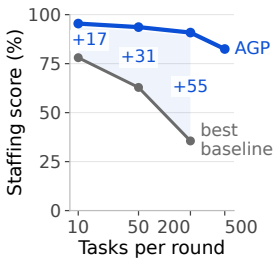


Figure 4: Scalability.

increasing action-space size from 10 to 500 on *staffing*, while keeping number of agents fixed. Figure 4 shows that AGP remains reasonably stable as this space expands: its score decreases from 95% at 10 tasks to 91% at 200 and 82% at 500, whereas DICG falls from 78% to 36% at 200, widening the gap from 17 to 55 points. At 500 tasks, DICG exceeds our 48 GB GPU memory budget, while AGP remains trainable (see Appendix F). This highlights the scalability of the action-graph representation as the number of candidate actions grows. AGP avoids enumerating the joint action space and retains its structured action-level exchange, at the cost of increased attention and communication. Section 6 quantifies these costs and shows that the communication can be substantially compressed.

Qualitative Example. Figure 5 illustrates how the learned action graph evolves during focus fire on `10m_vs_11m`. At first contact, four agents attack enemy $e5$ while two attack $e7$; the strongest across-agent interactions concentrate on the corresponding attack candidates. As enemies are eliminated, these interactions shift toward the team's next targets, tracking the changing focus-fire pattern rather than a fixed relation between agents. Similar action-aligned structure appears beyond this episode: same-target preferences emerge on seven of twelve StarCraft scenarios, staffing nodes attend more strongly to teammates selecting the same task, and LBF nodes emphasize teammates that choose LOAD (see Appendix I). Next, we probe AGP's learned interactions systematically through interventions and architectural ablations.

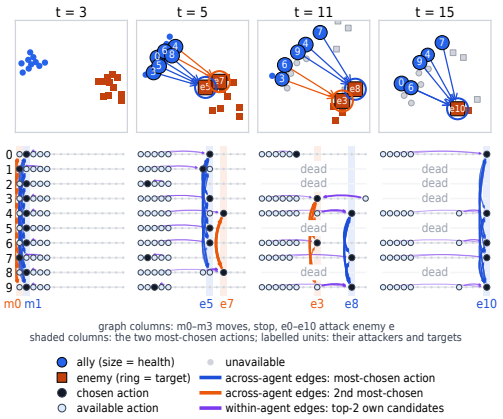


Figure 5: An AGP episode on `10m_vs_11m`.

6 Ablations and Discussion

We use the following analyses to identify what carries AGP's performance: which parts of the architecture are necessary during training, how much communication the representation actually requires, and whether trained policies rely on the learned action-level exchange. Also see Appendices G–J.

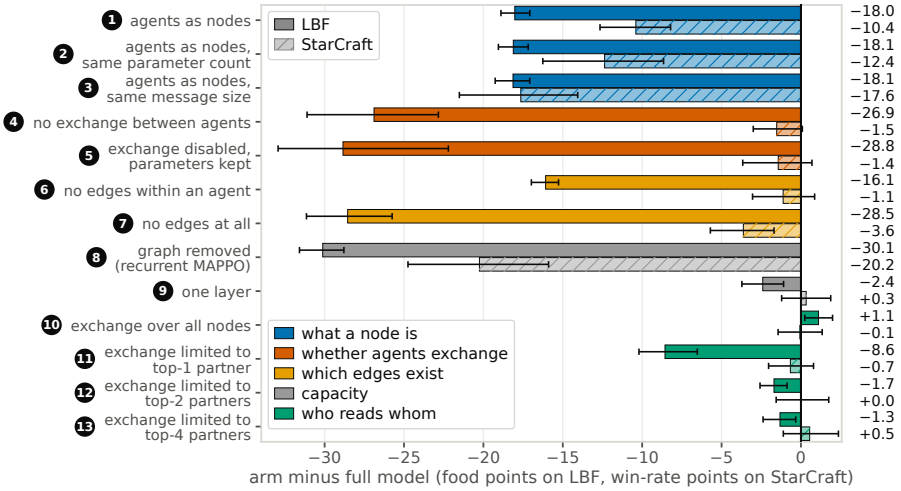


Figure 6: Training ablations relative to full AGP on LBF and StarCraft, with paired 95% confidence intervals; the numbered rows are cited in the text. Also see Appendix G.

Ablations. Figure 6 supports the main design choices behind AGP. Replacing candidate-action nodes with agent nodes substantially degrades both suites ❶, and matching AGP's parameter count or message size does not recover the loss ❷❸, suggesting that the action-level representation, rather than added capacity, drives the gain. On LBF, where success directly requires coordinated actions,

removing across-agent ❹ ❺ or within-agent ❻ interactions causes large losses, and removing both is more damaging ❼. Removing the graph altogether ❽ similarly causes large losses on both suites. StarCraft shows a more nuanced picture: individual graph ablations cause smaller but generally consistent losses across its diverse maps, while removing action-specific target features can be severe on larger and asymmetric maps (Figure 7). Since attack actions correspond to enemy targets, these features provide precisely the target context needed to distinguish and coordinate candidate actions. Finally, dense all-to-all attention offers little improvement ❿, while two layers ❾ and only a few matching partners ⓫–⓭ are generally sufficient, supporting AGP's sparse construction.

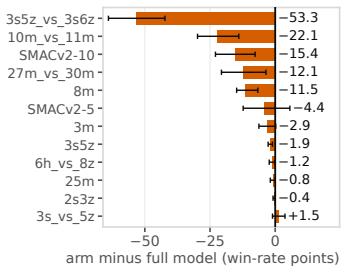


Figure 7: Removing target features on StarCraft, map by map.

Communication, computational cost and low-rank approximations. Computationally, our sparse action graph reduces $\mathcal{O}(N^2M^2d_x)$ dense attention to $\mathcal{O}(NM(N+M)d_x)$ in an AGP layer (Proposition 4.4; Appendix J). AGP maintains one representation per candidate action, giving each agent an $M \times d_x$ matrix to communicate per layer. Interestingly, this rich action-level representation need not imply an equally large message. We factorize the matrix into low rank-K action and content factors, reducing communication from Md_x to $K(M+d_x)$ values, and train the factorization end to end with the policy. As Figure 8 shows, small ranks closely preserve full-AGP performance, for instance, on the 200-task staffing game, rank 4 reduces the message from $25{,}728$ to $1{,}316$ values ($20\times$) with little observed loss. Thus, the benefit lies in maintaining action-specific structure, while the information exchanged over that structure can be highly compressed. Also see Appendix H.

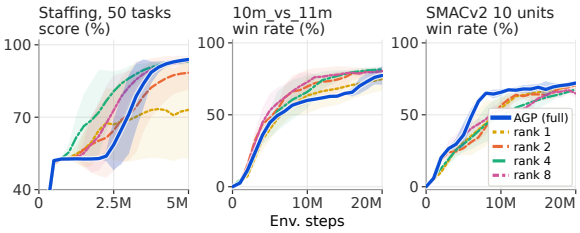


Figure 8: Low-rank approximation.

Post-hoc analysis. We also analyzed trained AGP policies via evaluation-time interventions: disrupting the learned attentions sharply degrades LBF and staffing performance and hurts most StarCraft maps. In LBF, AGP also exhibits fewer movement collisions than independent action draws and MAPPO. See Appendix I for per-task interventions, attention analyses, and behavioral examples.

Limitations. AGP currently assumes discrete actions. This covers many practical coordination problems, such as multi-robot task allocation (Gerkey & Matarić, 2004; Alirezazadeh & Alexandre, 2024), warehouse assignment (Zhen et al., 2025), UAV task or target selection (Alqudsi & Makaraci, 2025; Tahir et al., 2019), and coordinated robotic inspection (Ebert, 2022). Extending action graphs to continuous action spaces remains an important future direction. AGP also relies on communication at execution; a decentralized variant would broaden its applicability. Finally, maintaining representations per candidate action increases computation relative to agent-level policies. Low-rank messages reduce the communication cost, but reducing the computation itself would help under stricter constraints.

7 Conclusion

This paper proposes *Action Graph Policies*, which make candidate actions, rather than agents alone, the units of relational reasoning in cooperative MARL. AGP represents action co-dependence through a sparse, learned graph over agent-action pairs. Across 19 tasks spanning LBF, SMAC, SMACv2, and *Staffing*, AGP achieves the strongest aggregate performance against 14 other MARL methods. Its advantage is particularly clear in settings where successful behavior requires coordinated action choices, and it also remains effective when we scale up the action space to up to 500. More broadly, this work suggests that the choice of *what* a multi-agent policy reasons over is important. When outcomes depend on how possible actions interact, action graphs provide a more direct unit of reasoning. AGP offers a structured way to expose these dependencies without explicitly enumerating the joint action space, while retaining sparse connectivity. This perspective opens several directions for future work, including extending action-level reasoning to continuous action spaces, reducing communication and computation costs further, and deploying such policies in real-world multi-agent systems where communication, latency, and heterogeneous capabilities become central constraints.

AI USE STATEMENT

Large language models were used for grammar checking and refining the organization, presentation and overall quality of writing in this manuscript, and parts of the plotting scripts under the authors' direction. The authors manually reviewed and verified all material and take full responsibility for the final content of the paper. All research concepts, experimental design, implementations, analyses, and reported results were developed, conducted and validated by the authors. The conceptual framing, methodology, and scientific contributions of this work are entirely developed by the authors.

REPRODUCIBILITY STATEMENT

We share anonymized code at anonymous.4open.science/r/agp-marl: an implementation of AGP in the layout of the EPyMARL codebase (Papoudakis et al., 2021), run on the standard implementations of Level-Based Foraging (Christianos et al., 2020; Papoudakis et al., 2021), SMAC (Samvelyan et al., 2019) and SMACv2 (Ellis et al., 2023), together with our staffing game, implemented in the same codebase and defined in full in Appendix C. The codebase will be open-sourced if the paper is accepted. The main paper and the appendices give the details of the experiments: the AGP configuration, the training settings, the budgets and the seeds (Appendix E), the baselines (Appendix D), the ablation and message-budget arms (Appendices G and H) and the protocol of the post-hoc analyses (Appendix I).

REFERENCES

Saeid Alirezazadeh and Luís A Alexandre. A survey on task allocation and scheduling in robotic network systems. *IEEE Internet of Things Journal*, 12(2):1484–1508, 2024.

Yunes Alqudsi and Murat Makaraci. Uav swarms: research, challenges, and future directions. *Journal of Engineering and Applied Science*, 72(1):12, 2025.

Christopher Amato. An introduction to centralized training for decentralized execution in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2409.03052*, 2024.

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.

Navin Bhat and Kevin Leyton-Brown. Computing nash equilibria of action-graph games. *arXiv preprint arXiv:1207.4128*, 2012.

Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *International Conference on Machine Learning*, pp. 980–991. PMLR, 2020.

Jacopo Castellini, Frans A Oliehoek, Rahul Savani, and Shimon Whiteson. The representational capacity of action-value networks for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.07497*, 2019.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics, 2014.

Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *Advances in neural information processing systems*, 33: 10707–10717, 2020.

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pp. 746–752, USA, 1998. American Association for Artificial Intelligence. ISBN 0262510987.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *International Conference on machine learning*, pp. 1538–1546. PMLR, 2019.

Wei Duan, Jie Lu, and Junyu Xuan. Group-aware coordination graph for multi-agent reinforcement learning. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/434. URL https://doi.org/10.24963/ijcai.2024/434.

Julia Tenis Ebert. *Distributed Decision-making Algorithms for Inspection by Autonomous Robot Collectives*. Harvard University, 2022.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Nicolaus Foerster, and Shimon Whiteson. SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=5OjLGiJW3u.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 2974–2982, 2018.

Wei Fu, Chao Yu, Zelai Xu, Jiaqi Yang, and Yi Wu. Revisiting some common practices in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2206.07505*, 2022.

Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004.

Cong Guan, Feng Chen, Lei Yuan, Chenghe Wang, Hao Yin, Zongzhang Zhang, and Yang Yu. Efficient multi-agent communication via self-supervised information aggregation. *Advances in Neural Information Processing Systems*, 35:1020–1033, 2022.

Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pp. 227–234, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737.

Nikunj Gupta, James Zachary Hare, Jesse Milzman, Rajgopal Kannan, and Viktor Prasanna. Deep meta coordination graphs for multi-agent reinforcement learning. In *2026 International Conference on Advances in Artificial Intelligence and Machine Learning (AAIML)*, pp. 1046–1055, 2026. doi: 10.1109/AAIML67890.2026.11498169.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Diyi Hu, Chi Zhang, Viktor Prasanna, and Bhaskar Krishnamachari. Learning practical communication strategies in cooperative multi-agent reinforcement learning. In *Asian Conference on Machine Learning*, pp. 467–482. PMLR, 2023.

Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. Learning multi-agent communication from graph modeling perspective. In *International Conference on Learning Representations*, 2024.

Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. UPDeT: Universal multi-agent reinforcement learning via policy decoupling with transformers. *arXiv preprint arXiv:2101.08001*, 2021.

Albert Xin Jiang, Kevin Leyton-Brown, and Navin AR Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, 2011.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxdQkSYDB.

Yipeng Kang, Tonghan Wang, Qianlan Yang, Xiaoran Wu, and Chongjie Zhang. Non-linear coordination graphs. *Advances in neural information processing systems*, 35:25655–25666, 2022.

Russell Keith and Hung Manh La. Review of autonomous mobile robots for the warehouse environment. *arXiv preprint arXiv:2406.08333*, 2024.

Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(65):1789–1828, 2006. URL http://jmlr.org/papers/v7/kok06a.html.

Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pp. 764–772, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073.

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.

Yifan Liu, Dalei Wu, and Yu Liang. Survey on graph-based reinforcement learning for networked coordination and control. *Automation*, 6(4):65, 2025.

Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. Multi-agent game abstraction via graph attention neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7211–7218, 2020.

Zeyang Liu, Lipeng Wan, Xue Sui, Zhuoran Chen, Kewu Sun, and Xuguang Lan. Deep hierarchical communication graph in multi-agent reinforcement learning. In *IJCAI*, pp. 208–216, 2023.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Omayma Mahjoub, Sasha Abramowitz, Ruan De Kock, Wiem Khlifi, Simon Du Toit, Jemma Daniel, Louay Ben Nessir, Louise Beyers, Claude Formanek, Liam Clark, and Arnu Pretorius. Sable: a performant, efficient and scalable sequence model for marl. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Yaru Niu, Rohan Paleja, and Matthew Gombolay. Multi-agent graph-attention communication and teaming. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pp. 964–973, Richland, SC, 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450383073.

Frans A. Oliehoek and Chris Amato. A concise introduction to decentralized pomdps. In *SpringerBriefs in Intelligent Systems*, 2016. URL `https://api.semanticscholar.org/CorpusID:3263887`.

Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023.

Liviu Panait, Sean Luke, and R Paul Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

Alexander Peysakhovich and Adam Lerer. Prosocial learning agents solve generalized stag hunts better than selfish ones. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pp. 2043–2044, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020a.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020b.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pp. 2186–2188, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. In *International Conference on Learning Representations*, 2019.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pp. 5887–5896. PMLR, 2019.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2252–2260, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Chuxiong Sun, Zehua Zang, Jiabao Li, Jiangmeng Li, Xiao Xu, Rui Wang, and Changwen Zheng. T2mac: Targeted and trusted multi-agent communication through selective engagement and evidence-driven integration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 15154–15163, 2024.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Anam Tahir, Jari Böling, Mohammad-Hashem Haghbayan, Hannu T Toivonen, and Juha Plosila. Swarms of unmanned aerial vehicles—a survey. *Journal of Industrial Information Integration*, 16: 100106, 2019.

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

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020a.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. RODE: Learning roles to decompose multi-agent tasks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TTUVg6vkNjK.

Tonghan Wang, Liang Zeng, Weijun Dong, Qianlan Yang, Yang Yu, and Chongjie Zhang. Context-aware sparse deep coordination graphs. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=wQfgfb8VKTn.

Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. Action semantics network: Considering the effects of actions in multiagent systems. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=ryg48p4tPH.

Yutong Wang, Mehul Damani, Pamela Wang, Yuhong Cao, and Guillaume Sartoretti. Distributed reinforcement learning for robot teams: A review. *Current Robotics Reports*, 3(4):239–257, 2022b.

Muning Wen, Jakub Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. Multi-agent reinforcement learning is a sequence modeling problem. *Advances in Neural Information Processing Systems*, 35:16509–16521, 2022.

Maciej Wojtala, Bogusz Stefańczyk, Dominik Bogucki, Łukasz Lepak, Jakub Strykowski, and Paweł Wawrzyński. MACTAS: Self-attention-based module for inter-agent communication in multi-agent reinforcement learning. *arXiv preprint arXiv:2508.13661*, 2025.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

Qianlan Yang, Weijun Dong, Zhizhou Ren, Jianhao Wang, Tonghan Wang, and Chongjie Zhang. Self-organized polynomial-time coordination graphs. In *International conference on machine learning*, pp. 24963–24979. PMLR, 2022.

Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 5571–5580. PMLR, 2018.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

Lu Zhen, Zheyi Tan, René de Koster, Xueting He, Shuaian Wang, and Huiwen Wang. Optimizing warehouse operations with autonomous mobile robots. *Transportation Science*, 59(5):1130–1152, 2025.

APPENDIX

CONTENTS

A Extended Related Work

A.1 Problem setup and communication model

A.2 Execution regimes and families

B Formal Statements and Proofs

B.1 Sampling a co-dependent action pattern

B.2 Cost of higher-order action factors

B.3 Sparse global connectivity of the action graph

B.4 Receptive field of AGP

B.5 Computational cost

B.6 Properties of action graphs and action-graph policies

C Environments and Tasks

D Baselines

E Training, Evaluation and Compute

F Full Results

G Ablation Details

H Low-Rank Approximation

I Post-hoc Analyses of the Learned Graphs

I.1 Qualitative exhibits

J Computational Complexity

A EXTENDED RELATED WORK

This appendix expands Section 2. It first fixes the problem setup and the communication model, then states the three execution regimes by which the families differ, reviews the families in the order of how much crosses between agents at execution, and closes with the game-theoretic origin of the action graph.

A.1 PROBLEM SETUP AND COMMUNICATION MODEL

The environment is the Dec-POMDP of Section 2: agents observe locally, act simultaneously and share one reward. Communication is an augmentation of its information structure, not part of the environment; messages are neither actions nor rewarded, and the transition and observation functions do not depend on them. The setup is the following.

- *Local observation.* At step t agent i receives $o_{i,t}$ and maintains its history $\tau_{i,t}$; its history embedding $\mathbf{e}_i$ and its target features $\boldsymbol{\epsilon}_{i,a}$ are computed from this history alone (Eq. 6).
- *What is communicated.* In each of the L AGP layers, agent i broadcasts its current candidate-action representations $\{\hat{\mathbf{x}}_{i,a}^{(l)}\}_{a\in\mathcal{A}_i(s)}$, continuous vectors of width d_x, one per available action type, or their rank-K factorization (Appendix H). Nothing else crosses between agents: no observations, no histories and no sampled actions.

- *When.* The exchange happens within the decision step, in L synchronous exchanges, one per AGP layer, before any action is sampled; the across-agent update of Eq. 8 is the receiving end of layer l. Every agent samples its action after the last layer, and all agents act simultaneously. There is no one-step delay and no sequential commitment.
- *Learned.* The messages are intermediate representations of the actor, so they are learned end to end by the policy gradient of Section 4.2; there is no separate communication objective and no discrete communication action.
- *Information at action time.* Agent i chooses a_i from its own final representations $\{\mathbf{x}_{i,a}^{(L)}\}$ and its history score term (Eq. 10). By Proposition 4.3 these depend on the initial representations of the candidates in $\mathcal{F}_{i,a}^{(L)}$ and on nothing else.
- *What stays decentralized.* Each agent runs its own copy of the shared actor, computes its own updates from what it received, and samples its own action; no agent or controller sees the joint action before it is executed, and given the messages the joint policy is the product of Eq. 1.
- *What is unavailable at execution.* The global state s_t, other agents' observations and histories except through the messages, the critic, and the actions the others are about to take.

We assume a channel that connects every pair of agents within a decision step; Appendix J gives its bandwidth, LMd_x numbers per agent per step in full and $LK(M + d_x)$ at rank K.

This is the setting in which the communication and coordination-graph baselines of the comparison are stated and run. TarMAC (Das et al., 2019) describes its problem as a Dec-POMDP augmented with communication under centralized learning and decentralized execution, relaxed by allowing the agents to communicate; agents exchange continuous signature-and-value messages, received one step later, and act on local observations and received messages. CommFormer (Hu et al., 2024) states a Markov game over observations and adds a learned static communication graph under a bandwidth budget; agents receive teammates' observations and actions over it, and the authors describe execution as decentralized under a constrained channel. Deep coordination graphs (Böhmer et al., 2020) and their sparse variant CASEC (Wang et al., 2022a) state a Dec-POMDP and select the joint action by max-plus or max-sum message passing over a graph of agents at every step, exchanging value messages indexed by action for several rounds; both describe this as a distributed controller with a communication cost rather than as decentralized execution. AGP shares the model and the training regime with all four and differs in what travels over the channel, candidate-action representations instead of agent-level vectors or value messages, and in exchanging within the step rather than with a delay. Value-factorization and centralized-critic methods use the same Dec-POMDP with $\mathbf{m}_{-i} = \varnothing$, and autoregressive policies replace the channel by sequential commitment, as the execution regimes below make precise.

A.2 Execution regimes and families

Execution regimes. Every method in this paper is trained with access to the global state and to all agents' histories; the methods differ in what the *executed* policy may use. Under *independent execution* the joint policy is a product of local policies,

$$\pi(\mathbf{a} \mid \boldsymbol{\tau}) = \prod_{i=1}^{N} \pi_i(a_i \mid \tau_i), \tag{11}$$

which is Eq. 1 of Section 2 with $\mathbf{m}_{-i} = \varnothing$. Under *communicating execution* each agent j first emits a deterministic message $\mathbf{m}_j = g_j(\tau_j)$, and every agent then acts simultaneously from its own history and the messages it received, as in Eq. 1. There is no central controller: given the messages the joint policy is still a product, and sampling is parallel. This is the regime of AGP, whose messages are the candidate-action representations exchanged along the across-agent edges (Section 4.2). Under *sequential execution* agents commit one after another and each may condition on the actions already chosen, $\pi(\mathbf{a} \mid \boldsymbol{\tau}) = \prod_i \pi_i(a_i \mid \tau_i, a_{1:i-1})$.

Independent execution. Centralized training with decentralized execution (CTDE) (Kraemer & Banerjee, 2016; Amato, 2024) executes Eq. 11. Value-based methods learn a joint action value as a mixture of individual utilities under the Individual–Global–Max condition, so that each agent's greedy action is its component of the greedy joint action: VDN sums the utilities, QMIX mixes them

monotonically, and QTRAN, Weighted QMIX and QPLEX widen the class of representable joint values (Sunehag et al., 2017; Rashid et al., 2020b; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2020a). Policy-based methods such as MADDPG, COMA and MAPPO (Lowe et al., 2017; Foerster et al., 2018; Yu et al., 2022) train decentralized actors against a centralized critic. Action-level structure enters this family through an agent's own action space, in action semantics networks (Wang et al., 2020b), RODE (Wang et al., 2021) and UPDeT (Hu et al., 2021), which organize one agent's actions but represent no dependency among the candidate actions of different agents. In all of these, coordination is expressed only through the training signal.

Communicating execution. Communication methods execute Eq. 1. CommNet and TarMAC (Sukhbaatar et al., 2016; Das et al., 2019) share observation embeddings through averaging or attention; later work gates or targets who communicates (Singh et al., 2019; Niu et al., 2021; Sun et al., 2024), learns self-supervised or bandwidth-aware message contents (Guan et al., 2022; Hu et al., 2023), and routes messages through learned communication graphs or attention modules (Liu et al., 2023; Hu et al., 2024; Wojtala et al., 2025). Coordination graphs (Guestrin et al., 2002; Kok & Vlassis, 2006) keep simultaneous execution but structure the joint value over a graph of agents $\mathcal{G}_A = (\mathcal{N}, \mathcal{E}_A)$: Deep Coordination Graphs (DCG) (Böhmer et al., 2020) write Q_{tot} as a sum of unary payoffs $f_i(\tau_i, a_i)$ and pairwise payoffs $f_{ij}(\tau_i, \tau_j, a_i, a_j)$ over the edges of $\mathcal{G}_A$, as in Eq. 2, and select the joint action by message passing; variants learn the topology, aggregate payoffs non-linearly, or group and meta-learn the graph (Li et al., 2021; Wang et al., 2022a; Yang et al., 2022; Kang et al., 2022; Duan et al., 2024; Gupta et al., 2026), and graph-attention policies route information between agent nodes (Jiang et al., 2020; Liu et al., 2020; 2025). Two features matter for this paper: the payoffs are indexed by agents, so the coordination between two particular candidate actions is one entry learned implicitly inside an $|\mathcal{A}_i| \times |\mathcal{A}_j|$ table, and a factor over d agents has $|\mathcal{A}|^d$ entries, which is why deep coordination-graph methods stop at pairs (Proposition B.2). Non-linear coordination graphs (Kang et al., 2022) mix the payoffs non-linearly and are not covered by that additive bound.

Sequential execution. Autoregressive policies factorize the joint action by the chain rule; MAT and Sable (Wen et al., 2022; Mahjoub et al., 2025) realize this order at execution time, and AgentMixer exploits it during training before returning to independent execution (Li et al., 2025; Fu et al., 2022). The order is fully expressive, but agents commit one at a time, at the cost of N sequential decoding steps per decision (Appendix J).

Action graphs. That actions rather than agents can be the unit of a coordination structure is the idea behind Action-Graph Games (Jiang et al., 2011), which represent a game compactly on a graph over actions: the utility of choosing an action depends on the counts of agents choosing each action in its graph neighbourhood, not on which agents chose them, an anonymity that also underlies mean-field MARL (Yang et al., 2018). Action-graph games represent a given game with a specified graph and are solved by equilibrium computation (Bhat & Leyton-Brown, 2012). AGP brings the action graph to communicating execution: it learns the edge weights end to end and coordinates over the graph at execution time, keeping a representation for every candidate action rather than a count. Section 3 gives the argument for this choice and Appendix B the proofs behind it.

B Formal Statements and Proofs

This appendix states the two results behind Section 3, restates the formal claims of Section 4, and proves them; Section B.6 then collects the structural properties of action graphs and of the policies defined over them. Throughout, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the action graph of Definition 4.1, with $\mathcal{V} = \mathcal{N} \times \mathcal{A}$ and $\mathcal{E} = \mathcal{E}_{\text{within}} \cup \mathcal{E}_{\text{across}}$ as in Eq. 5, and $\mathcal{A}_j(s) \subseteq \mathcal{A}$ is the set of actions available to agent j in state s. The product policy of Proposition B.1 is Eq. 11 of Appendix A.

B.1 Sampling a co-dependent action pattern

Proposition B.1 (Sampling a co-dependent action pattern)**.** *Suppose a rewarding pattern, such as a co-dependence of order L in the sense of Definition 3.1, requires L agents to select prescribed actions, each with probability at most p under a product policy. The probability of sampling the complete pattern is at most p^L. Under uniform exploration over a shared action space $\mathcal{A}$, it is $|\mathcal{A}|^{-L}$.*

Proof. Let $C \subseteq \mathcal{N}$ with $|C| = L$ be the agents of the pattern and $a_i^\star$ the action prescribed to agent $i \in C$. Under a product policy (Eq. 11) the events $\{a_i = a_i^\star\}$ for $i \in C$ are independent given the joint history, so the probability that all of them occur is $\prod_{i \in C} \pi_i(a_i^\star \mid \tau_i) \le p^L$. The actions of the agents outside C are unconstrained and do not enter the product. Under uniform exploration every agent places probability $1/|\mathcal{A}|$ on each of its actions, so $p = 1/|\mathcal{A}|$ and the product equals $|\mathcal{A}|^{-L}$ exactly. □

Remark (Several rewarding patterns). If the reward at state s is positive exactly when one of K patterns occurs, each prescribing the actions of at least L agents, the union bound gives a probability of at most Kp^L of sampling any of them. In the foraging example of Section 3 the patterns are those in which a sufficient group of agents adjacent to an item all choose LOAD, and L is the size of the smallest sufficient group, two or three on our tasks depending on the levels drawn (Appendix C). The bound concerns the executed policy class and not any training algorithm: it says how rarely a product policy that has not yet learned the pattern will stumble on it.

B.2 COST OF HIGHER-ORDER ACTION FACTORS

Proposition B.2 (Cost of higher-order action factors)**.** *For a shared action space of size $|\mathcal{A}|$, an unrestricted payoff factor over L agents contains $|\mathcal{A}|^L$ entries. Including such a factor for every L-agent coalition requires $\binom{N}{L}|\mathcal{A}|^L$ entries. Moreover, for every $3 \le L \le N$ and $|\mathcal{A}| \ge 2$, there exist action co-dependences of order L in the sense of Definition 3.1, which no sum of unary and pairwise factors represents exactly.*

Proof. A payoff factor over a coalition C of L agents is a function $f_C(\boldsymbol{\tau}, \mathbf{a}_C)$ of the coalition's joint action $\mathbf{a}_C \in \prod_{i \in C} \mathcal{A}_i$, which for a shared action space of size $|\mathcal{A}|$ takes $|\mathcal{A}|^L$ values; a graph with a factor for every L-subset of $\mathcal{N}$ has $\binom{N}{L}$ of them.

For the last claim, take three agents i, j, k and the interaction $R(a_i, a_j, a_k) = \mathbf{1}\{a_i = a_i^\star,\ a_j = a_j^\star,\ a_k = a_k^\star\}$, which rewards one prescribed triple; for a larger coalition, fix the remaining members at their prescribed actions. Suppose that for some unary and pairwise functions

$$R(a_i, a_j, a_k) = f_i(a_i) + f_j(a_j) + f_k(a_k) + f_{ij}(a_i, a_j) + f_{ik}(a_i, a_k) + f_{jk}(a_j, a_k).$$

Pick $\bar{a}_i \ne a_i^\star$, $\bar{a}_j \ne a_j^\star$ and $\bar{a}_k \ne a_k^\star$, and form the alternating sum over the eight assignments in which each of the three agents is at its prescribed or its alternative action, with sign $+1$ when an even number of agents deviate and -1 otherwise:

$$\begin{aligned} D \ = \ & R(a_i^\star, a_j^\star, a_k^\star) - R(\bar{a}_i, a_j^\star, a_k^\star) - R(a_i^\star, \bar{a}_j, a_k^\star) - R(a_i^\star, a_j^\star, \bar{a}_k) \\ & + R(\bar{a}_i, \bar{a}_j, a_k^\star) + R(\bar{a}_i, a_j^\star, \bar{a}_k) + R(a_i^\star, \bar{a}_j, \bar{a}_k) - R(\bar{a}_i, \bar{a}_j, \bar{a}_k). \end{aligned}$$

Every unary and pairwise term contributes zero to D. A term that does not depend on agent k, say, takes the same value at the two assignments that differ only in a_k, and those two assignments enter D with opposite signs, so the pair cancels; the same holds for a term that does not depend on i or on j, and every term on the right-hand side omits at least one of the three agents. Hence $D = 0$ for any sum of unary and pairwise factors. For the interaction itself only the first assignment is rewarded, so $D = 1$, a contradiction. In the terms of Definition 3.1: under a sum of unary and pairwise factors, Δ_i is a sum of one term per teammate, so if changing the action of any proper subset of a coalition C alone leaves Δ_i unchanged, so does changing all of C, and every co-dependence has order two. The prescribed L-tuple is a co-dependence of order L: from the assignment in which the other $L - 1$ agents deviate to the one in which they all take their prescribed actions, Δ_i changes from 0 to 1, while any proper subset returning to its prescribed actions leaves it at 0. □

Remark (Two distinct limitations of coordination graphs). The proposition separates two problems that are easily conflated. The first is the cost of raising the factor order explicitly. With N agents and M actions each, a coordination graph that admits arbitrary k-agent interactions has $\binom{N}{k}$ candidate factors, and an unrestricted k-agent factor is a table over M^k joint configurations, so the representation alone can grow as $\binom{N}{k}M^k$ before any inference is run: with $N = 8$ and $M = 6$, the pairwise level has 28 factors of 36 entries, the ternary level 56 factors of 216 entries, and the four-way level 70 factors of 1,296 entries. The count is that of a fully general factor set at one order; a particular construction may share parameters across factors or restrict which coalitions appear, and max-plus or

max-sum inference adds its own cost on top. The second problem is what is lost by staying pairwise. The alternating sum in the proof shows that a genuine three-way interaction, one whose value depends on all three choices at once, has a nonzero third-order difference D, while every sum of unary and pairwise factors has $D = 0$, so no such sum represents it, whatever values the factors take. The three loaders of Eq. 4 and the exact staffing requirements of the staffing game (Appendix C) are interactions of this kind. Restricting a coordination graph to pairwise factors is thus an expressivity limitation for genuinely higher-order dependencies; raising the factor order removes it, at a combinatorial growth in the number and the size of the factors.

B.3 Sparse global connectivity of the action graph

Proposition 4.2. *For $N, M \geq 2$, the full, unmasked action graph has NM nodes and $|\mathcal{E}| = NM(N + M - 2)$ edges, compared with $\mathcal{O}(N^2M^2)$ for a fully connected graph over the same NM nodes. Its topology is the Cartesian product $K_N \,\square\, K_M$, where K_N and K_M are complete graphs over the N agents and M actions, respectively, and any two candidate-action nodes are connected by a path of length at most two.*

Proof. The node count is $|\mathcal{N} \times \mathcal{A}| = NM$.

Topology. The Cartesian product $K_N \,\square\, K_M$ of two complete graphs has vertex set $\mathcal{N} \times \mathcal{A}$, and two vertices (i, a) and (j, b) are adjacent if and only if either $i = j$ and ab is an edge of K_M, or $a = b$ and ij is an edge of K_N. Since every pair of distinct vertices is adjacent in a complete graph, this reads: adjacent if and only if ($i = j$ and $a \neq b$) or ($a = b$ and $i \neq j$). The first case is exactly $\mathcal{E}_{\text{within}}$ and the second exactly $\mathcal{E}_{\text{across}}$, so the undirected form of the action graph coincides with the product. This product is also known as the rook's graph on an $N \times M$ board.

Edge count. Fix a node (i, a). It has $M - 1$ out-neighbours in $\mathcal{E}_{\text{within}}$, namely (i, b) for $b \neq a$, and $N - 1$ out-neighbours in $\mathcal{E}_{\text{across}}$, namely (j, a) for $j \neq i$. The two sets are disjoint, because a within-agent neighbour has the same agent index and an across-agent neighbour a different one. Summing over the NM nodes gives $|\mathcal{E}| = NM(M - 1 + N - 1) = NM(N + M - 2)$. A fully connected directed graph over the same nodes has $NM(NM-1) = \mathcal{O}(N^2M^2)$ edges, so the action graph keeps a fraction $(N + M - 2)/(NM - 1)$ of them.

Diameter. Let (i, a) and (j, b) be distinct nodes. If $i = j$ or $a = b$ they are adjacent. Otherwise $i \neq j$ and $a \neq b$, and the path $(i, a) \to (i, b) \to (j, b)$ uses one within-agent edge and one across-agent edge, so their distance is at most two. Since $N, M \geq 2$, a pair with $i \neq j$ and $a \neq b$ exists and is not adjacent, so the diameter is exactly two. □

B.4 Receptive field of AGP

Proposition 4.3. *For an available candidate (i, a), define $\mathcal{F}_{i,a}^{(0)} = \{(i, a)\}$ and, for $l \geq 0$,*

$$\mathcal{F}_{i,a}^{(l+1)} = \bigcup_{j\colon a \in \mathcal{A}_j(s)} \; \bigcup_{b \in \mathcal{A}_j(s)} \mathcal{F}_{j,b}^{(l)}.$$

Then $\mathcal{F}_{i,a}^{(L)}$ contains exactly the candidates that can influence (i, a) through L graph layers. If $\mathcal{A}_i(s) = \mathcal{A}$ for every agent, then $\mathcal{F}_{i,a}^{(L)} = \mathcal{N} \times \mathcal{A}$ for all $L \geq 1$. More generally, one action available to every agent suffices for every available candidate to have a global receptive field for $L \geq 2$.

Proof. We show by induction on l that the representation $\mathbf{x}_{i,a}^{(l)}$ is a function of the initial representations $\{\mathbf{x}_{j,b}^{(0)} : (j, b) \in \mathcal{F}_{i,a}^{(l)}\}$ and of no other node. For $l = 0$ this is the definition of $\mathcal{F}_{i,a}^{(0)}$.

One AGP layer applies Eq. 7 and then Eq. 8, and unavailable candidates are masked out of the keys and values of both updates. In the within-agent update, $\hat{\mathbf{x}}_{j,b}^{(l)}$ attends over agent j's own available candidates, so it is a function of $\{\mathbf{x}_{j,b'}^{(l)} : b' \in \mathcal{A}_j(s)\}$; the residual connection adds $\mathbf{x}_{j,b}^{(l)}$ itself, which is in that set. In the across-agent update, $\mathbf{x}_{i,a}^{(l+1)}$ attends over the candidates (j, a) of the agents j for which action a is available, so it is a function of $\{\hat{\mathbf{x}}_{j,a}^{(l)} : j \text{ with } a \in \mathcal{A}_j(s)\}$; the residual

connection adds $\hat{\mathbf{x}}_{i,a}^{(l)}$, which is in that set because $a \in \mathcal{A}_i(s)$ by assumption. Composing the two, $\mathbf{x}_{i,a}^{(l+1)}$ is a function of $\{\mathbf{x}_{j,b}^{(l)} : a \in \mathcal{A}_j(s),\ b \in \mathcal{A}_j(s)\}$, and by the induction hypothesis each $\mathbf{x}_{j,b}^{(l)}$ is a function of the initial representations in $\mathcal{F}_{j,b}^{(l)}$. Hence $\mathbf{x}_{i,a}^{(l+1)}$ is a function of the initial representations in $\bigcup_{j:a\in\mathcal{A}_j(s)} \bigcup_{b\in\mathcal{A}_j(s)} \mathcal{F}_{j,b}^{(l)} = \mathcal{F}_{i,a}^{(l+1)}$, and of no other node, which is the claim. Whether a given candidate in the set actually influences $\mathbf{x}_{i,a}^{(L)}$ is decided by the learned attention weights; the proposition bounds where information can flow.

If every agent has the same available set $\mathcal{A}$, the outer union ranges over all $j \in \mathcal{N}$ and the inner over all $b \in \mathcal{A}$, so $\mathcal{F}_{i,a}^{(1)} = \bigcup_{j\in\mathcal{N}} \bigcup_{b\in\mathcal{A}} \{(j, b)\} = \mathcal{N} \times \mathcal{A}$.

The recursion is monotone, $\mathcal{F}_{i,a}^{(l)} \subseteq \mathcal{F}_{i,a}^{(l+1)}$, because the term $j = i$, $b = a$ of the union is always present, so a receptive field that is global at one layer stays global at every later layer, which gives the claim for every $L \geq 1$. If some action c is available to every agent, then for every agent j the outer union defining $\mathcal{F}_{j,c}^{(1)}$ ranges over all agents and the inner over all of their available actions, so $\mathcal{F}_{j,c}^{(1)}$ is the set of all available candidates. For an available candidate (i, a), the union defining $\mathcal{F}_{i,a}^{(2)}$ contains the term $j = i$, $b = c$, since a and c are both available to agent i, so $\mathcal{F}_{i,a}^{(2)} \supseteq \mathcal{F}_{i,c}^{(1)}$ is the set of all available candidates as well. □

B.5 Computational cost

Proposition 4.4. *One AGP layer in Eqs. 7–8 requires $\mathcal{O}\big(NM(N + M)\, d_x\big)$ attention operations, compared with $\mathcal{O}(N^2M^2d_x)$ for full attention over all NM action nodes.*

Proof. Multi-head attention over a set of n tokens of width d forms $n \times n$ query–key scores and the corresponding weighted sum of values, at a cost of $\mathcal{O}(n^2d)$; the per-token linear projections cost $\mathcal{O}(nd^2)$ and are identical for every scheme considered, so we count attention operations only.

The within-agent update (Eq. 7) runs one attention per agent over that agent's M candidates, for a total of $N \cdot \mathcal{O}(M^2d_x) = \mathcal{O}(NM^2d_x)$. The across-agent update (Eq. 8) runs one attention per action type over the N candidates that share it, for a total of $M \cdot \mathcal{O}(N^2d_x) = \mathcal{O}(MN^2d_x)$. Summing the two gives $\mathcal{O}\big(NM(N + M)\, d_x\big)$. Full attention treats all NM nodes as one set, at a cost of $\mathcal{O}\big((NM)^2d_x\big) = \mathcal{O}(N^2M^2d_x)$. The ratio of the two is $\Theta\big(NM/(N + M)\big)$, which grows without bound when both N and M grow and approaches $\min(N, M)$ when one of them is fixed, about eight on the staffing games; Appendix J gives the counts on our tasks. □

B.6 Properties of action graphs and action-graph policies

This section collects the structural properties that Section 4 uses, separating properties of the action graph, which hold for any policy defined over it, from properties of the AGP updates. Throughout, $\mathcal{A}_i(s)$ is the set of actions available to agent i at state s, and the *available graph* $\mathcal{G}(s)$ is the subgraph of $\mathcal{G}$ induced by the available candidates $\{(i, a) : a \in \mathcal{A}_i(s)\}$, the graph on which the masked updates of Eqs. 7–8 operate.

Properties of the action graph. *Size and shape.* By Proposition 4.2, $\mathcal{G}$ has NM nodes and $NM(N + M - 2)$ directed edges, its undirected form is the Cartesian product $K_N \,\square\, K_M$, every node has degree $N + M - 2$, and any two nodes are joined by a path of length at most two: a node reaches the nodes of its own agent and of its own action type in one step, and every other node in two, through the node that shares the agent of one and the action type of the other. *Action identity.* The two edge types are not interchangeable. Within-agent edges join different action types of one agent and across-agent edges join the same action type of different agents, so the graph records which action type a candidate stands for, and a candidate exchanges information with teammates only through candidates of the same type. *Connectivity under availability.* Restricting to available candidates keeps each agent's own candidates connected, since the within-agent edges join all of them, but connects two agents only where they share an available action. Call agents i and j adjacent when $\mathcal{A}_i(s) \cap \mathcal{A}_j(s) \neq \varnothing$. Two available candidates (i, a) and (j, b) then lie in the same component

of $\mathcal{G}(s)$ exactly when i and j are connected through a chain of adjacent agents, and only in that case can information pass between them, under any number of layers.

Properties of the AGP updates. *Receptive field.* Proposition 4.3 gives the exact set $\mathcal{F}_{i,a}^{(L)}$ of candidates that can influence (i, a) after L layers. Unfolding the recursion, a candidate of agent k lies in $\mathcal{F}_{i,a}^{(L)}$ exactly when there is a chain of agents $i = j_0, j_1, \ldots, j_l = k$ with $l \leq L$ in which j_0 and j_1 share the action a itself and every later consecutive pair shares some available action; the first link is fixed by the action type of the receiving candidate, the later links are free. The receptive field therefore grows with depth along chains of overlapping action sets, covers the whole component of (i, a) in $\mathcal{G}(s)$ once L is at least the largest such chain length, and is all of $\mathcal{N} \times \mathcal{A}$ after one layer when every agent has the same available set. *Order invariance.* The across-agent update of Eq. 8 is an attention over the set $\{\hat{\mathbf{x}}_{j,a}^{(l)} : a \in \mathcal{A}_j(s)\}$ of matching candidates. Attention is invariant to the order of its keys and values, so the update depends on this set as a multiset and not on which agent contributed which element. The within-agent update of Eq. 7 likewise does not depend on the order in which an agent's candidates are listed, beyond permuting its outputs in the same way, and distinguishes them only through their representations, which carry the action-type embedding $\mathbf{u}_a$ and the target features. *Equivariance to agent relabeling.* Let σ be a permutation of $\mathcal{N}$ and write $\sigma \cdot (\boldsymbol{\tau}, \boldsymbol{\epsilon})$ for the histories and target features with the agents relabeled by σ. Because every parameter of the actor is shared across agents and both updates are order-invariant, the candidate representations satisfy $\mathbf{x}_{\sigma(i),a}^{(l)}(\sigma \cdot (\boldsymbol{\tau}, \boldsymbol{\epsilon})) = \mathbf{x}_{i,a}^{(l)}(\boldsymbol{\tau}, \boldsymbol{\epsilon})$ at every layer, by induction on l from Eq. 6, and so does the policy of Eq. 10: relabeling the agents permutes the per-agent policies in the same way. The statement assumes that agent identity enters only through the observations; when an observation contains an agent identifier, the relabeling must permute the identifiers as well for the statement to hold. This is an analogue of the anonymity of action-graph games (Jiang et al., 2011), in which the utility of an action depends on how many agents choose each neighbouring action and not on which agents do; here the aggregation over matching candidates is learned rather than a count, and teammates remain distinguishable through their features and histories. *Cost.* Proposition 4.4 bounds one layer by $\mathcal{O}(NM(N + M)d_x)$ attention operations against $\mathcal{O}(N^2M^2d_x)$ for attention over all candidates, and Appendix J works through the constants.

Heterogeneous and disjoint action sets. *Alignment as a type map.* An agent whose native action set differs from the vocabulary enters the action graph through an injective map ϕ_i from its native actions to $\mathcal{A}$, which assigns each native action a type: actions of different agents that are the same, or that correspond semantically, receive the same type, and distinct actions of one agent receive distinct types. The correspondence is supplied by the application, for instance through the entity an action acts on or the task it selects, and is not given by equality of local action numbering. The type decides which candidates exchange information; which action an agent executes is decided by its policy over its own available candidates. Aligned actions of different agents share the type embedding $\mathbf{u}_a$ of Eq. 6 and are distinguished by the agent's history $\mathbf{e}_i$ and by the target features. In every task of this paper ϕ_i is the identity on a vocabulary shared by all agents, with attack actions aligned by the enemy they name and staffing actions by the task. *Size of the available graph.* With $m_i = |\mathcal{A}_i(s)|$ available candidates for agent i and n_a agents for which type a is available, $\mathcal{G}(s)$ has $\sum_i m_i$ nodes and $\sum_i m_i(m_i - 1) + \sum_a n_a(n_a - 1)$ directed edges, which is $NM(N + M - 2)$ when every agent has every type; masking can lengthen paths or split $\mathcal{G}(s)$ into components, and Proposition 4.3 reasons over exactly this graph. *Chains and depth.* With agents adjacent as defined above, along a chain of d adjacent pairs every candidate of the agent at one end reaches every candidate of the agent at the other within $d + 1$ layers: d across-agent steps carry the information along the shared types and one further within-agent step delivers it to the receiving agent's other candidates. A type available to every agent makes every pair adjacent, which is the two-layer case of Proposition 4.3, and two agents with no chain between them exchange nothing at any depth. A shared action serves as such a bridge whether or not any agent selects it, since only its availability is needed for its candidate to take part in both updates; an unavailable candidate cannot serve, because it is masked out of the keys and values. *Self-loops.* Both attention updates include the query candidate among their keys, and the residual path keeps its own representation, whereas Definition 4.1 lists no self-loops. This is why the recursion of Eq. 9 contains the term $j = i$, $b = a$ and is monotone in l. *Cost on the available graph.* If attention is evaluated only over the available groups, the scores and the weighted sums of one layer cost $\mathcal{O}\big(\big[\sum_i m_i^2 + \sum_a n_a^2\big]d_x\big)$ and the projections $\mathcal{O}\big(\sum_i m_i\, d_x^2\big)$, against $\mathcal{O}\big(\big[\sum_i m_i\big]^2 d_x\big)$

for attention over all available candidates. Our implementation pads every team to the full $N \times M$ array and masks, so it pays the full-array bound of Proposition 4.4 and Appendix J whatever the availability; the group bound is what an implementation that packs the available candidates would pay.

C Environments and Tasks

This appendix describes the three benchmark suites and the staffing game introduced in Section 5, and Table 1 lists every scenario with its number of agents N, its number of action types M and its training budget. Appendix E gives the training and evaluation protocol.

Level-Based Foraging. Level-Based Foraging (LBF) (Christianos et al., 2020; Papoudakis et al., 2021) is a grid world with N agents and a number of food items. Every agent and every item has a level. Each agent has six actions, NO-OP, the four moves and LOAD, so $M = 6$ on every foraging task. An item is collected in a step when the agents adjacent to it that choose LOAD in that step have levels summing to at least the item's level (Eq. 4); the team then receives a reward proportional to the item's level, and a LOAD that does not complete a collection has no effect. Two or more agents that move into the same empty cell in one step are all held in place, which is the collision rule of the post-hoc analysis in Appendix I. Our five tasks are the forced-cooperation variants: agent levels are drawn from $\{1, 2\}$ at the start of every episode and every item's level is set to the sum of the three lowest agent levels. With three agents, every item therefore needs all three to load it in the same step. With four to six agents, three loaders always suffice and two suffice only for level draws in which the three lowest levels sum to at most four, so which agents form the loading group is itself part of the coordination problem. Agents observe every agent and item on the grid, episodes last 100 steps, and the metric is the fraction of the total food level collected, in percent. The five tasks are 15x15-3p-5f, 15x15-4p-5f, 20x20-4p-8f, 20x20-5p-8f and 25x25-6p-10f, named by grid size, number of agents and number of items; the post-hoc analyses of Appendix I additionally use three forced-cooperation tasks with three agents on which AGP was also trained, 20x20-3p-6f, 25x25-3p-8f and a 15x15-3p-5f variant in which agents see only two cells around them.

StarCraft. This suite comprises the StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019) and SMACv2 (Ellis et al., 2023). Each agent controls one unit against a scripted enemy team. Its actions are the four moves, stop, and one attack action per enemy unit, so M is five plus the number of enemies, and the attack action with index k names the same enemy for every agent, which is what aligns the action types across agents. Each unit observes the allies and enemies within its sight range, the reward is the standard shaped reward for damage dealt, kills and winning the episode, and the metric is the win rate over evaluation episodes, in percent. For an attack action the target features $\boldsymbol{\epsilon}_{i,a}$ of Eq. 6 are the observed features of the named enemy (its relative position, health, shield and unit type), zero when that enemy is out of sight; the move and stop actions have no target features. The ten SMAC maps span homogeneous marine teams (3m, 8m, 25m), mixed stalker and zealot teams (2s3z, 3s5z), and asymmetric maps in which the team is outnumbered or faces a different unit type (3s_vs_5z, 3s5z_vs_3s6z, 10m_vs_11m, 6h_vs_8z, 27m_vs_30m). In the two SMACv2 scenarios, with five and ten units per side, the unit types of both teams are drawn every episode from the Protoss units (stalkers, zealots and colossi) in fixed proportions and the start positions are randomized, so the enemy that an attack action names changes from episode to episode.

The staffing game. The staffing game is a coordination game of our own that isolates two things: action co-dependence of an order above two (Definition 3.1), and the size of the action space. There are $N = 8$ agents and T tasks, and every agent has $M = T + 1$ actions, action 0 meaning abstain and action $t \in \{1, \ldots, T\}$ meaning choosing task t. An episode is 32 independent rounds, each drawn as follows.

- Four of the T tasks are drawn uniformly without replacement and are *live* for the round; the other tasks pay nothing.
- The live tasks receive the staffing requirements $(3, 2, 2, 1)$ in a uniformly random order; the requirements sum to N, so a joint action that staffs every live task exactly exists in every round and uses every agent.
- Each live task receives a value v_t drawn uniformly from $[0, 1]$.

- Each agent sees each live task independently with probability one half. For every action type t the agent observes three numbers, whether task t is visible to it, the visible value v_t and the visible requirement $r_t/3$, all zero for tasks it does not see and for tasks that are not live; the agent may choose a task it does not see. These per-action numbers are the target features $\boldsymbol{\epsilon}_{i,t}$ of AGP's candidate actions and, concatenated over t, the observation of every baseline.

With c_t the number of agents that choose task t, the round's reward, shared by all agents, is

$$\mathcal{R} = \frac{\sum_{t \text{ live}} v_t \cdot \max\bigl(0,\ 1 - |c_t - r_t|/r_t\bigr)}{\sum_{t \text{ live}} v_t}, \tag{12}$$

which equals one exactly when every live task is staffed by exactly its requirement and gives partial credit for near misses (the values are positive with probability one, so the ratio is defined). Under Definition 3.1, an agent's choice of task t is co-dependent on the coalition of r_t teammates whose joining takes the task from under- to over-staffed, a co-dependence of order $r_t + 1$, so of order three and four on the tasks with $r_t \geq 2$. The metric is the mean round reward, the fraction of the attainable value, in percent; the post-hoc analysis in Appendix I also uses the indicator that all four live tasks are staffed exactly. Rounds are independent, so the discount is zero (Appendix E). The reward itself is not a sum of pairwise payoffs: for a task with requirement two, the third difference of Eq. 12 over three agents that choose between that task and abstaining, with every other agent abstaining, equals $-v_t / \sum_{t' \text{ live}} v_{t'}$, and for requirement three, with one further agent already on the task, it equals $-\frac{2}{3}\, v_t / \sum_{t' \text{ live}} v_{t'}$, so no sum of unary and pairwise factors represents it exactly (Proposition B.2); this is the structure the game is built to expose. Scenarios with $T = 10$, 50, 200 and 500 tasks are used: the ten- and fifty-task games enter the main comparison of Section 5, on which every baseline was run, and the two larger games form the scaling study, with the number of candidate actions per agent growing from 11 to 501 while everything else stays fixed.

Table 1: Every scenario of the paper: number of agents N, number of action types M per agent, and the training budget in environment steps. The action vocabulary is shared by all agents of a scenario.

Suite	Scenario	N	M	Budget
Staffing	10 tasks	8	11	5M
	50 tasks	8	51	5M
	200 tasks (scaling)	8	201	5M
	500 tasks (scaling)	8	501	5M
Foraging	15x15-3p-5f	3	6	50M
	15x15-4p-5f	4	6	50M
	20x20-4p-8f	4	6	50M
	20x20-5p-8f	5	6	50M
	25x25-6p-10f	6	6	50M
SMAC	3m (3 marines vs. 3)	3	8	20M
	2s3z (2 stalkers, 3 zealots vs. same)	5	10	20M
	3s5z (3 stalkers, 5 zealots vs. same)	8	13	20M
	3s_vs_5z (3 stalkers vs. 5 zealots)	3	10	20M
	3s5z_vs_3s6z (3 stalkers, 5 zealots vs. 3 stalkers, 6 zealots)	8	14	20M
	8m (8 marines vs. 8)	8	13	20M
	10m_vs_11m (10 marines vs. 11)	10	16	20M
	6h_vs_8z (6 hydralisks vs. 8 zealots)	6	13	20M
	25m (25 marines vs. 25)	25	30	20M
	27m_vs_30m (27 marines vs. 30)	27	35	20M
SMACv2	5 units (Protoss, types drawn per episode)	5	10	20M
	10 units (Protoss, types drawn per episode)	10	15	20M

D BASELINES

This appendix describes the fourteen methods that Section 5 compares AGP with, grouped by the families of Section 2 and by what crosses between agents when the trained policy is executed. Table 2 summarizes them; Appendix E gives the shared training protocol, and Appendix J compares the cost of AGP with that of each family.

Communication. *TarMAC* (Das et al., 2019) lets every agent broadcast a signature and a value vector each step; a receiver weights the incoming values by the match between its own query and the senders' signatures, over several rounds, so what crosses is an attention-weighted summary of what the teammates observed. *MASIA* (Guan et al., 2022) consolidates all received messages into one representation by attention and trains that representation with a self-supervised reconstruction objective in addition to the task reward. *T2MAC* (Sun et al., 2024) lets each agent select which teammates to consult and integrates their messages with an evidential weighting that discounts unreliable senders. *CommFormer* (Hu et al., 2024) learns a sparse $N \times N$ communication graph end to end on a transformer backbone; the graph decides which agents' encodings are combined, and, as in the original implementation, actions are decoded in sequence as in MAT, so this method also has a sequential element at execution. *MAGIC* (Niu et al., 2021) uses a learned scheduler to produce a communication graph each step and processes messages with graph attention over several rounds. *GA-Comm* (Liu et al., 2020) combines a hard attention that decides who communicates with whom and a soft graph attention that decides how much each message counts. In all six, the unit that is sent and received is one vector per agent.

Coordination graphs. *DCG* (Böhmer et al., 2020) writes the joint action value as a sum of unary payoffs and pairwise payoffs over a complete graph of agents (Eq. 2), with the payoffs computed from the agents' recurrent embeddings, and selects the joint action by max-plus message passing over that graph at execution. *DICG* (Li et al., 2021) learns the graph implicitly: attention over the agents' embeddings produces edge weights and a graph convolution over them produces the per-agent values that are then mixed; we use its value-based form. *CASEC* (Wang et al., 2022a) keeps DCG's pairwise payoffs but selects a sparse set of edges each step by a variance-based criterion, so that message passing runs over a context-dependent graph. What crosses between agents in this family is the payoff messages of the graph inference, indexed by agent pairs.

Independent execution. *QMIX* (Rashid et al., 2020b) mixes the agents' individual action values monotonically into a joint value during training and executes each agent's greedy action alone. *MAPPO* (Yu et al., 2022) trains recurrent per-agent actors with PPO against a centralized critic on the global state; it is also the training scheme of AGP, so the pair isolates the actor (Section 4.2). *RODE* (Wang et al., 2021) decomposes the action space into role-specific subspaces from learned action representations and lets each agent first pick a role and then an action within it; our implementation places its role selector and role-conditioned action subspaces on the recurrent MAPPO actor, so that it shares the training protocol of the other policy-based systems. Nothing crosses between agents at execution in this family.

Sequential execution. *MAT* (Wen et al., 2022) encodes all agents' observations with a transformer and decodes their actions one agent at a time, each conditioned on the actions already chosen. *Sable* (Mahjoub et al., 2025) keeps this autoregressive decoding and replaces attention over time by retention, giving it memory across steps. What crosses between agents is the sequence of committed actions.

E Training, Evaluation and Compute

This appendix gives the protocol behind every number in the paper: the AGP configuration and the optimization scheme of Section 4.2, the budgets and seeds of Section 5, the evaluation schedule and the estimator, what AGP communicates at execution, and the hardware. Appendix C describes the scenarios and Appendix D the baselines.

AGP configuration. The history encoder is a GRU with a 128-dimensional hidden state, the candidate-action representations have width $d_x = 128$, the action-type embeddings $\mathbf{u}_a$ have width 32, and the target features are projected to width 32 before the concatenation of Eq. 6. There are $L = 2$ AGP layers, each a within-agent update followed by an across-agent update (Eqs. 7–8) and a position-wise feed-forward network of width 256, with four attention heads, pre-layer normalization and residual connections. The policy is Eq. 10, with the history score term $\mathbf{W}_b\mathbf{e}_i$ computed from the GRU state. The centralized critic is a recurrent network on the global state. The target features $\boldsymbol{\epsilon}_{i,a}$ are the observed features of the named enemy on StarCraft and the visible value and requirement of

Table 2: The fourteen baselines by family and by what crosses between agents when the trained policy acts. AGP belongs to the communicating regime of Appendix A: what crosses is the set of candidate-action representations exchanged along the across-agent edges.

Method	Family	What crosses between agents at execution
TarMAC	communication	attention-weighted messages, one vector per agent
MASIA	communication	messages consolidated by attention into one representation
T2MAC	communication	messages from selected teammates, evidentially weighted
CommFormer	communication	encodings over a learned sparse agent graph; actions decoded in sequence
MAGIC	communication	messages over a scheduled graph with graph attention
GA-Comm	communication	messages gated by hard and weighted by soft attention
CASEC	coordination graph	pairwise payoff messages over a sparse, context-selected agent graph
DCG	coordination graph	pairwise payoff messages over the complete agent graph
DICG	coordination graph	agent embeddings over an implicitly learned agent graph
QMIX	independent	nothing
MAPPO	independent	nothing
RODE	independent	nothing
MAT	sequential	the actions already chosen
Sable	sequential	the actions already chosen

the named task on the staffing game; foraging actions have no target and the feature is empty there (Appendix C). All actor parameters are shared across agents.

Optimization. All policy-based systems, AGP included, are trained with PPO (Schulman et al., 2017) in the centralized-training scheme of MAPPO (Yu et al., 2022): rollouts of 128 steps, four epochs of two minibatches per rollout, learning rate 2.5×10^{-4} for actor and critic, clipping $\varepsilon = 0.2$, entropy coefficient 0.01, generalized advantage estimation with $\lambda = 0.95$, discount 0.99 on the benchmark suites and 0 on the staffing game whose rounds are independent, and gradient-norm clipping at 0.5. With advantages $\hat{A}_t$, each agent contributes the clipped surrogate

$$\mathcal{L}_i(\theta) = \mathbb{E}_t\Big[\min\big(\rho_{i,t}(\theta)\,\hat{A}_t,\ \mathrm{clip}(\rho_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon)\,\hat{A}_t\big)\Big], \qquad \rho_{i,t}(\theta) = \frac{\pi_i^{\theta}(a_{i,t} \mid \tau_{i,t}, \mathbf{m}_{-i,t})}{\pi_i^{\theta_{\text{old}}}(a_{i,t} \mid \tau_{i,t}, \mathbf{m}_{-i,t})}, \tag{13}$$

to the actor objective $\sum_i \mathcal{L}_i(\theta)$ plus the entropy bonus, and the critic minimizes a clipped value loss on the global state. For AGP the messages $\mathbf{m}_{-i,t}$ are computed from the other agents' histories through the same parameters, so the gradient of agent i's objective flows through the across-agent edges into every other agent's encoder and candidate-action representations; there is no auxiliary communication loss. Value-based baselines use their own published training rule with a replay buffer and epsilon-greedy exploration.

Budgets and statistical significance. Budgets are 50M environment steps on foraging, 20M on StarCraft and SMACv2, and 5M on the staffing game. Each experiment is run with five seeds for statistical significance, and the reported score of a system on a scenario is the mean over seeds.

Evaluation and estimator. Every run is evaluated at fixed intervals over its budget, 20 times on StarCraft and the staffing game and 50 times on foraging, each evaluation being 32 episodes of the greedy or sampled policy according to the system's convention, with an additional evaluation of the untrained policy at step zero so that all curves share an origin. The reported score of a run is the mean of the evaluation metric over the evaluations in the last tenth of the budget; a scenario score is the mean over seeds; a suite score is the equal-weight mean over the suite's scenarios. Intervals are 95% bootstrap intervals over seeds with 2,000 draws, paired by scenario when two systems are compared.

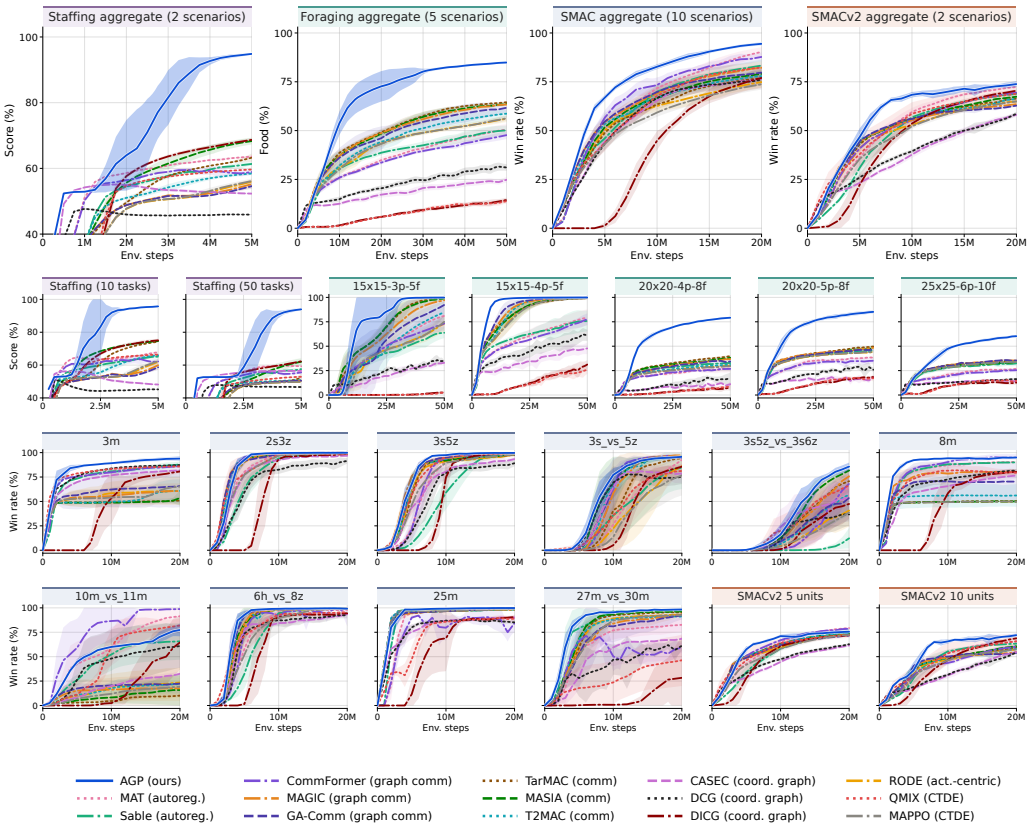


Figure 9: Learning curves on every scenario. Top row: the four suite aggregates of Figure 3. Rows two to four: the nineteen scenarios, the two staffing games and the five foraging tasks, then the ten SMAC maps, then the two SMACv2 scenarios, tinted by suite. Curves are means of the training-episode metric with 95% bands; the legend gives each baseline's family.

What AGP communicates at execution. Each agent runs its own GRU and builds its own M candidate-action representations from local information (Eq. 6). The within-agent update of Eq. 7 is local to the agent. The across-agent update of Eq. 8 needs communication: in each of the L layers every agent broadcasts its M current candidate-action representations, Md_x numbers, over a decentralized channel, and each candidate attends to the matching candidates it received. After the last layer every agent samples independently from Eq. 10, in parallel. There is no global controller, no access to the state and no enumeration of the joint action space; the per-step cost is L layers of Proposition 4.4 plus one policy head, and Appendix H gives the low-rank form of the broadcast.

Compute. All experiments were conducted on 48 GB NVIDIA RTX 6000 Ada Generation GPUs.

F Full Results

This appendix gives the learning curve of every system on every scenario behind the main comparison of Section 5, a description of what those curves show, the probability that AGP improves on each baseline over all nineteen scenarios, and the action-space scaling study in full. The estimator behind every number is that of Appendix E.

Learning performance. Figure 9 shows the four suite aggregates of Figure 3 together with the nineteen scenario curves they average. Three things stand out across the panels. First, where the reward depends on several agents choosing compatible actions in the same step, AGP learns earlier and finishes higher: on both staffing games it separates from every baseline early in training and climbs to 95 and 94 percent of the attainable value, and on the five foraging tasks it reaches 84 percent

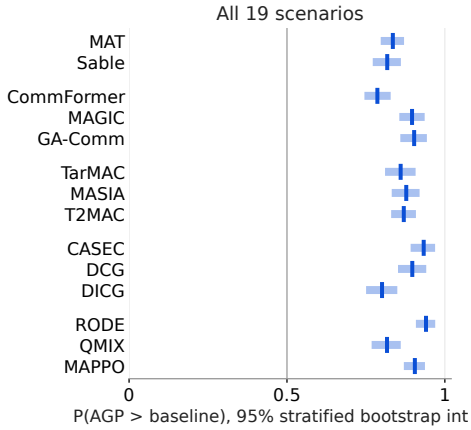


Figure 10: Probability that an AGP run scores above a baseline run on the same scenario, pooled over all nineteen scenarios with equal weight per scenario, with 95% stratified bootstrap intervals; ties count one half. Every interval lies above one half.

of the food where the best baseline, TarMAC, reaches 64, with the largest margins on the three large grids where the loading groups have to assemble across more agents and more items. Second, on the StarCraft maps, where units already coordinate reasonably well through what they observe, the margin is smaller but the curve is steeper: AGP reaches the final suite score of the strongest baseline, CommFormer, after 14.8M of the 20M steps and finishes at 94 percent wins against 88. Third, on the two SMACv2 scenarios, whose unit types are redrawn every episode, AGP learns as fast as the strongest baselines and finishes among them.

Consistency across scenarios. The per-scenario panels explain why AGP leads the aggregates by more than any single scenario would suggest. Many baselines are strong on some scenarios and weak on others: TarMAC reaches 99.6 percent wins on 2s3z and 25m but 49 on 3m and 10 on 10m_vs_11m; MASIA follows the same pattern; CommFormer, the best baseline on SMAC, is far behind on foraging and the staffing games; and DICG, the best baseline on SMACv2 and on the staffing games, is among the weakest on foraging. AGP's curves are instead at or near the top of nearly every panel. It is first on all seven staffing and foraging scenarios and on five of the ten SMAC maps, within its interval of the first system on four more, and behind on one map, 10m_vs_11m, where CommFormer finishes at 99 percent and AGP at 74. The aggregate lead is therefore a matter of consistency: no baseline is competitive on every kind of task, and AGP is.

Scale and variety of the scenarios. The nineteen scenarios differ in what they ask of a team. The staffing games reward exact coalition sizes on a task set that is mostly out of view; the foraging tasks reward groups of two or three agents, out of teams of three to six, loading the same item at the same moment on grids from 15×15 to 25×25 cells; the StarCraft maps span homogeneous and mixed unit types, symmetric and outnumbered fights, and teams from three to twenty-seven units; and SMACv2 redraws the team every episode. Most reported action-level and graph-based methods are evaluated on teams of a handful of agents. The 25m and 27m_vs_30m maps, with 750 and 945 nodes in the action graph, and the fifty-task staffing game, with 408, show that AGP trains and coordinates at that size too: it reaches 99.8 and 97.7 percent wins on the two largest maps and 94 percent of the attainable value on the fifty-task game. Appendix C lists every scenario with its number of agents and candidate actions.

Probability of improvement. Figure 10 gives, for each baseline, the probability that a run of AGP scores above a run of that baseline on the same scenario, pooled over all nineteen scenarios. It lies between 0.79, against CommFormer, and 0.94, and the interval is above one half for every baseline. Over the same scenarios AGP's interquartile mean is 93.3 against 79.2 for the best baseline by that measure, MAT.

Scaling the action space. Table 3 gives the score of every system that was run on the staffing scaling axis, from ten to two hundred tasks per round, that is, from 11 to 201 candidate actions per agent and from 88 to 1,608 nodes in AGP's action graph, with the number of agents fixed at eight. AGP's score stays high across this range, 95, 94 and 91 percent of the attainable value, and at five hundred tasks, where the graph has 4,008 nodes and each agent chooses among 501 candidates, it still reaches 82 percent (Figure 4). At five hundred tasks DICG did not fit in the 48 GB of GPU memory of Appendix E and could not be trained, so AGP is the only system with a score there. The action graph therefore grows by a factor of forty-five along this axis while the policy keeps most of its score, with no change to the architectural template, the message format or the training settings other than the minibatch count needed to fit memory. The baselines that reach two hundred tasks lose about half of their ten-task score over the same range. Section 6 and Appendix H show that the message AGP sends on these games can be compressed twentyfold at no observed cost, so the size of the exchange is not what limits scaling further.

Table 3: Action-space scaling on the staffing game at 5M steps: score in percent of the attainable value against the number of tasks per round, mean $\pm$ standard deviation over trials. The baselines shown are the strongest of their family on the ten- and fifty-task games; CommNet is not in the main roster and is included for completeness.

System	10 tasks	50 tasks	200 tasks
AGP (ours)	95.4 ± 0.5	93.6 ± 0.6	90.9 ± 2.8
DICG	78.0 ± 1.1	62.9 ± 0.1	35.5 ± 1.1
TarMAC	73.2 ± 0.3	50.7 ± 0.7	34.6 ± 1.7
MAT	67.3 ± 0.2	59.5 ± 0.1	33.9 ± 7.5
CommNet	67.6 ± 0.7	57.0 ± 0.5	36.1 ± 0.8
MAPPO	60.7 ± 1.4	49.9 ± 0.1	29.7 ± 2.7

G Ablation Details

Section 6 reports the ablations of Figure 6 in brief. This appendix gives the number behind each row on the two suites of that figure, the five foraging tasks and the twelve StarCraft scenarios, and then reports the same questions on the ten-task staffing game, which we ran as an additional investigation (Table 4). Every arm changes a single setting of the full model and is trained with the protocol and budget of Appendix E. The cost of an arm is its suite score minus that of full AGP, with a task-paired bootstrap interval: food points on foraging, win-rate points on StarCraft and score points on staffing.

What a node is. The first question is whether the unit of the exchange matters. *Agents as nodes* ❶ replaces the M candidate-action nodes of each agent by a single node built from the history embedding, so that the architecture exchanges one vector per agent, as the communication baselines of Appendix D do, and keeps everything else. This costs 18 food points on foraging and 10 win-rate points on StarCraft. Two matched controls rule out the obvious alternative explanations: *same parameter count* ❷ widens the single node until the control has as many parameters as AGP, and *same message size* ❸ gives it the payload of AGP's M nodes, so that it sends as many numbers. Both cost the same 18 points on foraging and more on StarCraft, 12 and 18 points. Neither capacity nor bandwidth substitutes for the candidate-action unit: the gain lies in representing each action separately.

Whether agents exchange, and along which edges. On foraging the exchange is decisive. *No exchange between agents* ❹ removes the across-agent edges, so that every agent reasons over its own candidates alone, and costs 27 food points; *exchange disabled, parameters kept* ❺ keeps the across-agent attention block and lets each node attend only to itself, so that the parameters and the compute of the block stay while no information crosses agents, and costs 29. What the block contributes is therefore the information it moves, not its parameters. *No edges within an agent* ❻ keeps the across-agent edges only, so that a candidate hears its teammates' matching candidates but cannot compare itself with the agent's own alternatives before the softmax, and costs 16 points; *no edges at all* ❼ removes both edge types and costs 28, about as much as removing the across-agent exchange alone. The within-agent edges thus pay off when there is something received to weigh the alternatives against, and both edge types are needed for the full effect. On StarCraft the same four

arms cost between one and four points, and the no-edges arm is clearly below the full model (its interval runs from -5.7 to -1.7), so the picture is the same at a smaller scale. Small margins are what this suite offers: units already coordinate well through what they observe, the full model wins 94 percent of its episodes, and the exchange still adds to that.

Removing the graph, and the capacity arms. *Graph removed* ❽ takes away the candidate-action nodes altogether and leaves a recurrent MAPPO actor. It costs 30 points on foraging, the same as removing every edge, and 20 points on StarCraft, far more than the four points of the no-edges arm. Most of the StarCraft gain over MAPPO therefore comes from the candidate-action nodes themselves, each carrying its action type and its target features, and the exchange adds a few points on top. *One layer* ❾ instead of two costs two food points and nothing on StarCraft, where one layer already lets an attack candidate read the matching candidates of every agent that has the same target in range (Proposition 4.3). One or two attention heads instead of four, and a node width of 64 instead of 128, change the score by at most a point and a half on either suite, so AGP is not a larger network winning on size. The history score term of Eq. 10 costs nothing on StarCraft, and on foraging the model without it matches full AGP on four tasks and never starts learning on the fifth, 15x15-3p-5f, which is why we keep the term.

Who reads whom. *Exchange over all nodes* ❿ lets every node attend to all NM nodes in one attention, a strict superset of the action graph's edges at $NM/(N+M)$ times more attention pairs (Proposition 4.4). It adds one food point on foraging and changes nothing on StarCraft, so the sparse graph gives up nothing to dense attention on these suites. In the other direction, *top-k partners* keeps only the k largest across-agent weights of each node. A single partner ⓫ is too few and costs 9 food points, while two or four partners ⓬ ⓭ cost one to two points and change StarCraft by less than a point: a candidate needs a few teammates' matching candidates, not all of them, which is also what the low-rank message of Appendix H exploits.

What a node knows. *Nodes without target features* keeps every node and every edge but sets $\boldsymbol{\epsilon}_{i,a} = \mathbf{0}$, so that a node knows which action it is but not what that action would act on. On StarCraft this costs 10 points on the suite, and Figure 7 shows where: 53 points on 3s5z_vs_3s6z, 22 on 10m_vs_11m, 15 on the ten-unit SMACv2 scenario and 12 on 27m_vs_30m, the large and asymmetric maps, while on the maps that most systems solve the action type alone is enough. This is the expected place for the effect. An attack action names an enemy, so the enemy's position, health and unit type are precisely the action-relevant information a candidate needs to tell a finishing blow from a wasted one, and every agent already observes them; the candidate-action node is where that information can be attached. Foraging actions have no target, so the question does not arise there.

Table 4: Ablations on the ten-task staffing game: arm minus full AGP in score points, with 95% bootstrap intervals. The rank-K message arms on this game are in Table 5.

arm	staffing, 10 tasks
agents as nodes	-38.3 [-42.9, -34.8]
nodes with the action type only	-31.6 [-32.3, -30.7]
nodes without the task's value and requirement	-24.1 [-33.1, -17.5]
no exchange between agents	-32.6 [-34.7, -29.8]
exchange over all nodes	-12.2 [-14.5, -9.3]
one layer	-6.8 [-7.1, -6.4]

The staffing game. We ran the arms that bear on the questions above on the ten-task staffing game as well, whose reward depends on exact coalition sizes (Appendix C); Table 4 gives the costs. The effects are the largest of any suite and point the same way. Agents as nodes costs 38 points, and removing the exchange costs 33: on a task whose reward is about how many teammates choose the same task, a candidate must be able to read the teammates' matching candidates. Dense attention over all nodes costs 12 points, so here the sparse structure helps rather than merely costing nothing: the column of matching candidates is exactly the set a task choice has to count. One layer instead of two costs 7 points, where the second layer lets the coalition sizes settle. The node arms show what a node needs to know: a node built from the action type alone costs 32 points, adding the agent's history recovers 8 of them, and adding the task's visible value and requirement, the target features of this game, recovers the remaining 24. Read with Figures 6 and 7, the three suites rank the arms in the

same order: what a node is and whether nodes exchange matter most, what a node knows matters wherever an action has a target, and capacity matters little.

H Low-Rank Approximation

This appendix defines the low-rank message of Section 6, states when it compresses the exchange, and gives its cost on every suite.

The rank-K message. At execution agent j broadcasts its M candidate-action representations after the within-agent update, the matrix $\hat{\mathbf{X}}_{j,:} \in \mathbb{R}^{M \times d_x}$, in each AGP layer (Appendix E). The rank-K arm replaces this broadcast by the least-squares rank-K factorization

$$\hat{\mathbf{X}}_{j,:} \approx \mathbf{U}_j \mathbf{V}_j^\top, \qquad \mathbf{U}_j \in \mathbb{R}^{M \times K},\ \mathbf{V}_j \in \mathbb{R}^{d_x \times K}, \tag{14}$$

that is, K loading vectors over the action types and K content vectors, so that $K(M + d_x)$ numbers travel instead of $M d_x$. Each receiver reconstructs the matrix and attends to it as in Eq. 8. The loading vectors are produced from the agent's own node matrix by a learned linear head, and the content vectors are the least-squares projection of the matrix onto their span with a small ridge term, so the factorization is a differentiable orthogonal projection that needs no singular-value decomposition and costs $\mathcal{O}(MKd_x + K^3)$ per agent. The arm is trained end to end with the factorization in place, so the policy learns to put what teammates need into the K components. A rank compresses the message only when $K < Md_x/(M + d_x)$, which with $d_x = 128$ is 5.7 on foraging ($M = 6$) and larger on every other scenario. Ranks 1, 2 and 4 therefore compress the message on every suite, and Table 5 reports them; higher ranks were also trained where they compress and behave like rank 4.

What the rank costs. Table 5 gives each rank's cost against the full model on every suite. Rank 4 costs nothing on foraging, on SMAC and on the ten-task staffing game, under a point on the fifty-task game, and on the two-hundred-task game it finishes nearly three points above the full model, which is itself still improving at the budget; on the two SMACv2 scenarios it is four points below the full model with an interval that spans zero, so that comparison is undecided. Rank 2 costs about a point on the benchmark suites and six points on the fifty-task staffing game, and is level with the full model on the two-hundred-task game. Rank 1 is where the compression begins to bite: it costs seven food points on foraging, driven by a single task, three win-rate points on SMAC, and sixteen and twenty-two points on the ten- and fifty-task staffing games, where some runs never reach the full model, yet nothing at two hundred tasks, where every restricted arm is level with or above the full model. The cost of rank 1 is therefore not monotone in the size of the action space and is reported per scenario. The picture is consistent across the benchmark suites and the staffing games: the exchange saturates by rank 4 on the tested suites, and a message of a few components carries what the candidates need.

Table 5: Rank-K message arms, arm minus full AGP with 95% bootstrap intervals, per suite.

Suite	Rank 1	Rank 2	Rank 4
Foraging (5)	-7.1 $[-20.0, +0.1]$	-0.7 $[-2.0, +0.6]$	-0.1 $[-1.0, +0.8]$
SMAC (10)	-2.7 $[-6.1, +0.2]$	-1.0 $[-3.0, +0.7]$	-0.0 $[-1.5, +1.4]$
SMACv2 (2)	-2.2 $[-10.0, +5.6]$	$+0.1$ $[-6.7, +7.8]$	-3.8 $[-10.8, +4.1]$
Staffing, 10 tasks	-15.8 $[-42.0, -2.1]$	-0.8 $[-1.2, -0.3]$	$+0.0$ $[-0.5, +0.6]$
Staffing, 50 tasks	-22.4 $[-40.8, -6.9]$	-5.9 $[-13.7, -0.9]$	-0.6 $[-1.4, +0.2]$
Staffing, 200 tasks	-0.3 $[-3.3, +2.6]$	$+1.5$ $[-1.3, +4.3]$	$+2.8$ $[+0.0, +5.4]$

What the rank saves. On the two-hundred-task staffing game the full message is $201 \times 128 = 25{,}728$ numbers per agent per layer and the rank-4 message $4 \times (201 + 128) = 1{,}316$, a twentyfold reduction at no observed cost; on the StarCraft maps the full message is between one and four and a half thousand numbers and rank 4 holds it between 540 and 660; on foraging it is 768 against 536. At rank 4 the exchange is therefore four to five times a single 128-number agent-level message on the benchmark suites and ten times on the two-hundred-task game, which is the comparison the agent-node control of Appendix G makes in the other direction: the same message size with one node per agent costs 18 food points. The structure of the exchange, one candidate reading the matching candidates of its teammates, is what carries the result, and the content of that exchange can be small.

Where the restriction helps. On 10m_vs_11m the rank-2 and rank-4 arms finish above the full model, and on the fifty- and two-hundred-task staffing games the rank-4 arm learns faster than the full model through the middle of training. These are the places where restricting the message may act as a regularizer, an effect that low-rank payoff factorizations also show in coordination graphs (Böhmer et al., 2020); the intervals there are wide.

I Post-hoc Analyses of the Learned Graphs

This appendix gives the analyses that Section 6 summarizes and the qualitative exhibits of Section 5 in full: what happens when the learned across-agent weights are replaced at evaluation time, how much weight a candidate places on teammates choosing the same action, how the trained policy avoids collisions on foraging, and two episodes walked through step by step, all on the scenarios of the main comparison.

Protocol. Every analysis reads trained AGP policies after the fact, on the scenarios of the main comparison: the final checkpoints on the twelve StarCraft scenarios, the five foraging tasks and the ten-task staffing game (Appendix C), rolled out with the sampling policy for 32 episodes with the attention weights, actions and environment state recorded at every step. The attention weights are descriptive; the interventions are the one analysis with a causal reading, and they measure what a trained policy relies on, not what a policy needs to reach its score.

Interventions. The across-agent weights are replaced at evaluation time in two ways: set to uniform over the available senders, so that every candidate still receives its teammates' matching candidates but without selectivity, or the exchange is removed, so that each node attends only to itself. Figure 11 shows the win rate under the three conditions on every StarCraft scenario and the food collected on every foraging task. On seven of the twelve StarCraft scenarios uniform weights keep at least 0.94 of the trained win rate, on the two SMACv2 scenarios they have a minor effect, and on three maps they cost: 10m_vs_11m ($0.84 \to 0.62$), 3s5z_vs_3s6z ($0.65 \to 0.38$) and 27m_vs_30m ($0.97 \to 0.26$). Removing the exchange costs on ten of the twelve, from a few points to a collapse: 3s5z_vs_3s6z to zero, 27m_vs_30m to 0.06, 6h_vs_8z to 0.12, 10m_vs_11m to 0.15. On the five foraging tasks the food collected falls from 79 percent to 12 under uniform weights and to 23 without the exchange, and on the staffing game the fraction of rounds in which every live task is staffed exactly falls from 0.77 to 0.02 and to 0.00, the level of the agent-node control. The trained policies use the exchange everywhere, and on most StarCraft maps its presence more than its selectivity.

Same-choice weights. For a node with action type a, the senders are split into those whose chosen action is a and the rest, and the weight on each group is expressed relative to uniform, at the second AGP layer, with the node itself excluded. On the staffing game a node on a live task reads the teammates that chose that task at 1.41 times uniform and the others at 0.79, a ratio of 1.8. On StarCraft a preference for same-choice senders is present on seven of the twelve scenarios, largest on 25m (1.72 against 0.87), 27m_vs_30m and 10m_vs_11m, absent on 3m and 8m, and slightly reversed on 3s_vs_5z, 6h_vs_8z and the five-unit SMACv2 scenario. On foraging the preference exists on the LOAD action alone, where a loading candidate reads the teammates that are loading at 1.15 times the weight it gives the others with three agents and nearly twice with six, and on the other action types the structure is a column rather than a diagonal: whatever action a node stands for, it reads the teammates that chose to load. The preference is thus strongest where the coordination is about matching, on the staffing game and the large maps, and takes a different but equally task-shaped form on foraging.

Collision avoidance. Foraging holds every agent in place when two or more agents choose moves that land on the same empty cell. Figure 12 gives the per-step rate of such conflicts under the trained policy, under the rate that independent draws from each agent's move frequencies, pooled over the rollout and applied to the same steps, would produce, and under the two interventions. On the two 15×15 tasks the trained policy conflicts at 0.15 and 0.22 of its independence rate, and on the three larger grids, where conflicts are rarer, at 0.37 to 0.70. On 15x15-4p-5f, where AGP, MAPPO and the agent-node control all collect 99 to 100 percent of the food, AGP conflicts at 0.042 per step against 0.111 for MAPPO and 0.088 for the control. Relative to its own independence rate the conflict ratio rises from 0.15 to 0.56 under uniform weights and to 1.78 without the exchange on 15x15-3p-5f, so the avoidance depends on the exchange. It is expressed in the joint decision rather than in a visible

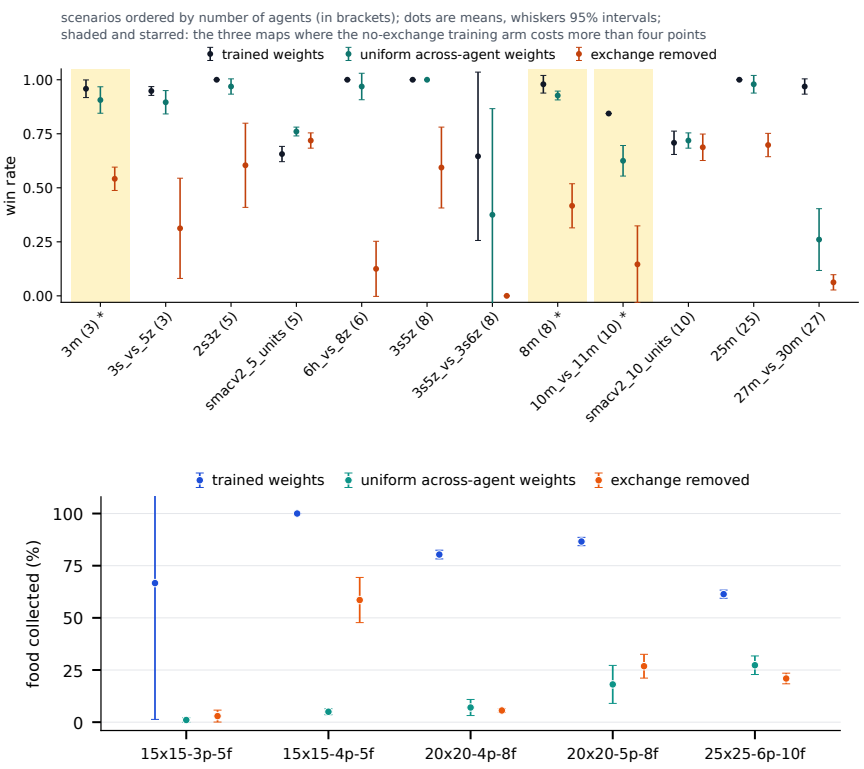


Figure 11: Evaluation-time interventions on the across-agent weights. Top: win rate per StarCraft scenario under the trained weights, uniform weights and no exchange, with 95% intervals, scenarios ordered by agent count; the three shaded maps are those where the no-exchange training arm costs more than four points. Bottom: the same on the five foraging tasks, in percent of the food collected.

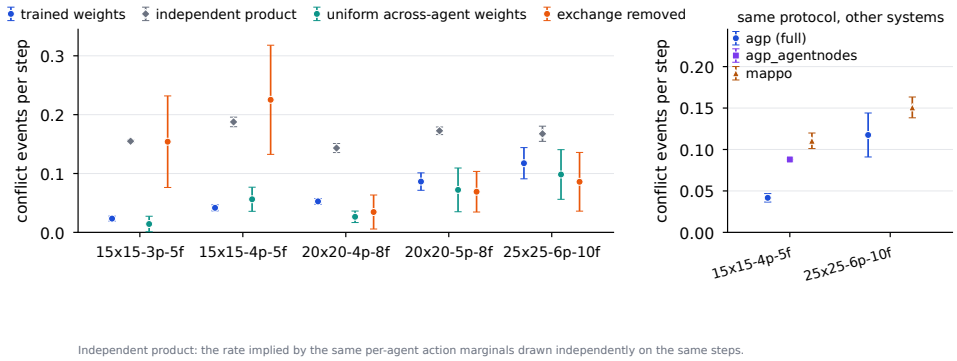


Figure 12: Same-cell move conflicts per step on foraging. Left: per task, under the trained policy, under independent draws from each agent's move frequencies pooled over the rollout and applied to the same steps, and under the two interventions, with 95% intervals. Right: the three systems that reach the full score on 15x15-4p-5f under the same protocol, and the same comparison on 25x25-6p-10f where MAPPO scores half of AGP.

edge: an agent that could contend for a cell takes that move with probability 0.11 when a teammate's chosen move contends for it and 0.24 when none does, where independent draws would give 0.26 in both cases, while the weight it puts on that teammate is no larger than on the others. Nothing in the reward names collisions; the policy avoids them because the exchange lets each candidate move read the teammates' candidate representations, which carry their propensities, before anyone commits.

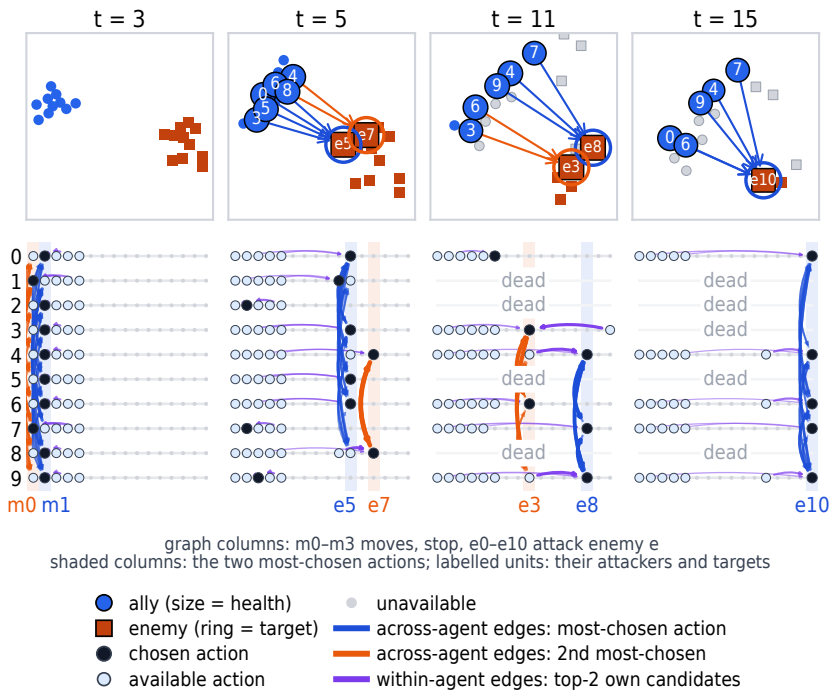


Figure 13: The episode of Figure 5 at full size: ten marines against eleven on 10m_vs_11m at four steps. The team moves together, opens fire in two groups on two targets, and converges on the remaining enemies as its own losses mount.

I.1 Qualitative exhibits

How an exhibit is read. Figures 13 and 14 show one won episode each at four steps. The top row is the battlefield: allies as blue circles whose size follows their health, enemies as orange squares, dead units in grey, and an arrow from each ally to the enemy it attacks; a ring marks an enemy attacked by several allies. The bottom row is the action graph at the same step, drawn as the $N \times M$ array of Section 4.1: one row per agent, one column per action type, the four moves and stop followed by one attack action per enemy. Filled nodes are the chosen actions, open nodes the available ones, small grey dots unavailable ones, and a dead agent's row is greyed out. Blue arcs are the across-agent edges on the column of the most-chosen attack action, orange arcs those on the second most-chosen column, drawn between the agents that choose them with width proportional to the attention weight, and purple arcs are the two strongest within-agent edges of every chosen node, pointing at the candidates of the same agent that it reads most. The episodes are selected by a fixed rule: a won episode, at least two enemy targets under attack by at least four attackers at some step, and, among those, the step at which the across-agent weights are least explained by a graph with one weight per pair of agents, so that the per-action structure is at its most visible. They are chosen to be readable, not to be typical, and the statistics above are the evidence for how common each pattern is.

Ten marines against eleven. Figure 13 is the episode of Figure 5 at full size. At $t = 3$, before contact, eight agents choose the same move and two another, and the across-agent weight sits on the two movement columns: the team moves as one block, and the candidates that carry weight are the moves the teammates are choosing, not the attacks, which are not yet available. At $t = 5$ the teams meet. Four agents attack enemy $e5$ and two attack $e7$, and the across-agent edges that carry weight are those between the agents converging on the same target: the $e5$ column joins the four attackers of $e5$, the $e7$ column joins the other two, and each group leaves the other's column almost empty. The purple arcs show that each chosen attack node reads the agent's own movement candidates most, the alternatives it is choosing against. By $t = 11$ four agents have fallen; three survivors attack $e8$ and two attack $e3$, and the weight has moved with them to the columns of the new targets. At $t = 15$ the five remaining agents all attack $e10$, and the $e10$ column joins every chosen node. The structure the framing predicts, that on the column of an action the senders who carry weight are the agents choosing it, is what this map shows at every step after contact.

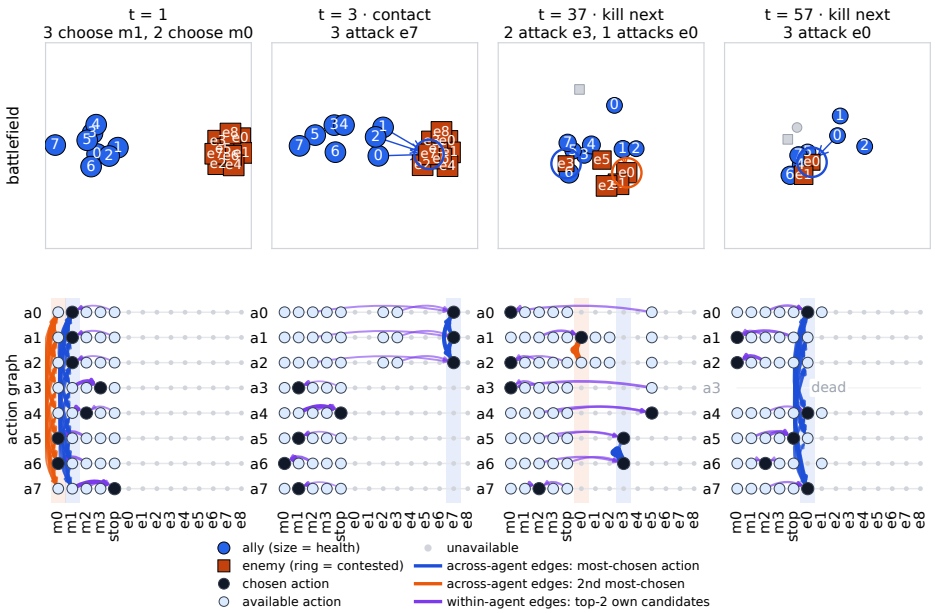


Figure 14: A second episode: three stalkers and five zealots against three stalkers and six zealots on 3s5z_vs_3s6z, won at $t = 65$, at four steps. Agents 0 to 2 are the stalkers, a ranged unit, and agents 3 to 7 the zealots, a melee unit.

Mixed units against a larger team. Figure 14 is a longer episode on an asymmetric map with two unit types, which lets the exhibit show a pattern the marine map cannot: the two unit types act at different times, and the graph follows them. At $t = 1$ the team is out of contact. Three agents choose one move and two another, and the across-agent weight lies on the two movement columns, as on the marine map; the attack columns are empty because no enemy is in range. At $t = 3$ the three stalkers, agents 0 to 2, are the first in range, and all three attack $e7$. The $e7$ column joins exactly these three nodes, and their within-agent edges reach back across the whole row to their movement candidates, the alternative of closing in that each stalker is choosing against. The zealots, agents 3 to 7, are still moving or holding position, and their chosen nodes read only their own neighbouring moves. At $t = 37$ the fight has broken into two engagements: two agents attack $e3$ and one attacks $e0$, with the across-agent weight on the $e3$ column joining the pair and the $e0$ column carrying the single attacker's read of the teammates that could join it; the other agents are moving or holding position, and the enemies that have fallen are marked in grey. At $t = 57$ one zealot is dead and the team has converged on $e0$: three agents attack it and the $e0$ column joins their chosen nodes, while two stalkers move away from the melee. The episode is won eight steps later. Two things are worth noting for a reader of Section 5. First, the graph is dynamic in the sense the main text claims: the same across-agent edges carry weight on movement columns before contact and on attack columns after it, and which attack column they carry weight on changes with the target. Second, the attention is not a record of who is attacking whom, since the arrows in the battlefield already show that; it is the pattern of which candidate actions of which teammates each candidate reads before the agents commit, and on both maps that pattern lines up with the group that ends up choosing the same action.

J Computational Complexity

This appendix works through the cost of AGP step by step: what one execution step computes and how that scales with the number of agents N, the number of action types M and the width d_x; how much memory training needs; how much the agents communicate; how many sequential stages a decision takes; and how each of these compares with the alternatives of Section 2. Throughout, L is the number of AGP layers and H the number of attention heads, with $L = 2$, $H = 4$ and $d_x = 128$ in every experiment (Appendix E). We count multiply-accumulate operations up to constants and write d for d_x.

Step 1: building the candidate-action representations. Each agent encodes its history with a GRU, at $\mathcal{O}(qd_e + d_e^2)$ per agent and step for an observation of width q, and forms its M initial representations by Eq. 6, a shared linear map from the concatenation of the history embedding, the action-type embedding and the target features to d dimensions. With $d_{\text{in}} = d_e + d_u + d_\epsilon$ this costs $\mathcal{O}(NM\, d_{\text{in}}\, d)$ in total. Every candidate is built from local information, so this stage is embarrassingly parallel across agents and needs no communication.

Step 2: the within-agent update. Eq. 7 runs one multi-head attention per agent over that agent's M candidates. For a set of n tokens of width d, attention costs $\mathcal{O}(nd^2)$ for the query, key, value and output projections and $\mathcal{O}(n^2 d)$ for the scores and the weighted sum, plus $\mathcal{O}(nd^2)$ for the position-wise feed-forward layer. With $n = M$ and N agents, the within-agent update costs

$$\mathcal{O}\big(NM\, d^2 + NM^2\, d\big). \tag{15}$$

It is local to each agent.

Step 3: the across-agent update. Eq. 8 runs one attention per action type over the N candidates that share it. With $n = N$ and M types this costs

$$\mathcal{O}\big(NM\, d^2 + N^2 M\, d\big). \tag{16}$$

This is the one stage that crosses agents: before it, every agent broadcasts its M current representations, and the attention of candidate (i, a) ranges over the received (j, a).

Step 4: one layer and the full policy. Adding Eqs. 15 and 16, one AGP layer costs

$$\mathcal{O}\big(\underbrace{NM(N+M)\, d}_{\text{attention}} + \underbrace{NM\, d^2}_{\text{projections}}\big), \tag{17}$$

which is Proposition 4.4 with the projection term made explicit. The policy head of Eq. 10 adds $\mathcal{O}(NMd + NMd_e)$. The whole execution step is therefore $\mathcal{O}\big(L\, NM\, ((N+M)\, d + d^2)\big)$, with a sequential depth of L layers. Two regimes follow from Eq. 17. When $N + M < d$, which holds on every benchmark scenario ($N + M \leq 62$), the projection term dominates and the cost is linear in the number of nodes NM: AGP then costs about $NM/N = M$ times a per-agent policy of the same width, the price of keeping one representation per candidate rather than one per agent. When $N + M > d$, the attention term grows past the projection term; on the 200- and 500-task staffing games the two are of the same order, and the attention term grows as $NM(N + M)$, still a factor $NM/(N + M)$ below attention over all NM nodes. Table 6 gives both terms on our scenarios.

Table 6: Per-layer cost of AGP on representative scenarios at $d_x = 128$: number of nodes, attention pairs formed by the within-agent and across-agent updates, attention pairs that dense attention over all nodes would form and the ratio between the two, the leading term NMd_x^2 of the projection cost in millions of multiply-accumulates, and the message each agent broadcasts per layer in numbers.

Scenario	N	M	NM	$NM(N+M)$	$(NM)^2$	ratio	$NMd_x^2/10^6$	Md_x
Foraging 15x15-4p-5f	4	6	24	240	576	2.4	0.4	768
Foraging 25x25-6p-10f	6	6	36	432	1,296	3.0	0.6	768
SMAC 3m	3	8	24	264	576	2.2	0.4	1,024
SMAC 8m	8	13	104	2,184	10,816	5.0	1.7	1,664
SMAC 10m_vs_11m	10	16	160	4,160	25,600	6.2	2.6	2,048
SMAC 25m	25	30	750	41,250	562,500	13.6	12.3	3,840
SMAC 27m_vs_30m	27	35	945	58,590	893,025	15.2	15.5	4,480
Staffing, 10 tasks	8	11	88	1,672	7,744	4.6	1.4	1,408
Staffing, 50 tasks	8	51	408	24,072	166,464	6.9	6.7	6,528
Staffing, 200 tasks	8	201	1,608	336,072	2,585,664	7.7	26.3	25,728
Staffing, 500 tasks	8	501	4,008	2,040,072	16,064,064	7.9	65.7	64,128

Step 5: memory. The parameter count of AGP does not grow with N and grows with M only through the action-type embeddings $\mathbf{u}_a$ and the history score matrix $\mathbf{W}_b$, both $\mathcal{O}(M)$; the attention and projection weights are shared by every node. The activations of one step are $\mathcal{O}(L\, NM\, d)$ for the representations and $\mathcal{O}(L\, H\, NM(N+M))$ for the attention maps when these are materialized. Training stores them for every environment in the batch and every step of the rollout, so with B parallel

environments and rollouts of T steps the activation memory of a training update is $\mathcal{O}\big(BTL\,(NMd + H\,NM(N+M))\big)$. For instance, on 27m_vs_30m, with 945 nodes, 64 environments and 128-step rollouts, the attention maps alone hold about 2×10^9 numbers per layer. The memory grows with NM.

Step 6: communication. In each layer every agent broadcasts its M representations, Md numbers, and receives $(N-1)Md$; over a step the channel carries $L\,NMd$ numbers. The rank-K message of Appendix H reduces the broadcast to $K(M+d)$ numbers, which on the two-hundred-task game is a twentyfold reduction at rank 4. Agent-level communication methods send one d-dimensional vector per agent per round, so AGP's full message is M times larger and its rank-K message $K(M+d)/d$ times larger, that is, $4 + M/32$ times at $K = 4$, between 4.2 and 5.1 on the foraging and StarCraft scenarios; the matched-bandwidth control of Appendix G shows that giving an agent-level design this bandwidth does not recover the gap. The operation counts above are for a batched computation of every agent's update; an agent that recomputes keys and values for the representations it receives repeats that projection once per received candidate.

Step 7: latency and sequential depth. Every candidate-action representation of a layer is computed in parallel, and every agent samples its action in parallel after the last layer, so a decision takes L layers of local computation and L broadcasts regardless of N and M. This is the same depth as a communication method with L exchange steps. An autoregressive policy decodes one agent after another and has depth N, and a coordination graph resolves its joint action by K_{mp} iterations of message passing over the agent graph, each of which costs $\mathcal{O}(|\mathcal{E}_A|\,M^2)$ for pairwise payoffs, $\mathcal{O}(N^2M^2)$ on a complete graph.

Table 7: Per-step cost of the executed policy by family, for N agents with M actions each, representation width d, L layers or exchange steps, and K_{mp} message-passing iterations. Depth is the number of sequential stages before all agents have acted.

Policy	Computation per step	Numbers sent per agent	Depth
Independent (MAPPO)	$\mathcal{O}(Nd^2)$	0	1
Agent-level communication, L exchange steps	$\mathcal{O}(L(Nd^2 + N^2d))$	Ld	L
Coordination graph, complete agent graph	$\mathcal{O}(Nd^2 + K_{\mathrm{mp}}N^2M^2)$	$K_{\mathrm{mp}}NM$	K_{mp}
Autoregressive transformer	$\mathcal{O}(N^2d + Nd^2)$, in N sequential steps	one action index per turn	N
Dense attention over NM candidates	$\mathcal{O}(L(NMd^2 + N^2M^2d))$	LMd	L
AGP	$\mathcal{O}(L(NMd^2 + NM(N+M)d))$	LMd, or $LK(M+d)$ at rank K	L

Comparison. Table 7 collects the per-step costs. Against dense attention over all NM candidate nodes, AGP saves a factor $NM/(N+M)$ in attention pairs, three on the six-agent foraging task, fifteen on 27m_vs_30m and eight on the 500-task staffing game, and the ablation of Appendix G shows that the dense arm buys nothing with the extra pairs. Against a coordination graph, whose pairwise payoffs cost $\mathcal{O}(N^2M^2)$ per message-passing iteration and whose payoff tables grow with M^2, AGP's cost grows with $NM(N+M)$ and its parameters grow with M only through the action-type embeddings and the history score matrix; AGP trains at 500 tasks, where the coordination-graph baseline did not fit in memory (Appendix F). Against an autoregressive transformer, whose encoder and decoder together cost $\mathcal{O}(N^2d + Nd^2)$ but whose decoder commits the agents one at a time, AGP does more arithmetic per step by roughly a factor M but finishes in L parallel exchanges rather than N sequential steps. Against agent-level communication, AGP's cost is a factor M larger in the projection term and its message a factor M larger, the price of representing every candidate action; the ablations attribute the gains to exactly that representation.

What this costs in practice. The constants matter. On the benchmark scenarios AGP's step is dominated by NM projections of width d, that is, by running a transformer block over a few hundred nodes at every environment step, which makes AGP more expensive to train than an agent-level policy of the same width by a constant factor. The analysis above says where that cost comes from and how it scales: linearly in the number of candidate actions on the benchmark suites, and with $NM(N+M)$ once the team or the action space is large, with the message compressible by an order of magnitude and the decision depth fixed at L.