MARL-GPT: Foundation Model for Multi-Agent Reinforcement Learning

AAAI Track

Maria Nesterova
AXXX & MIRAI
Moscow, Russia
minesterova@yandex.ru

Mikhail Kolosov
MIRAI
Moscow, Russia
mikek7532@gmail.com

Anton Andreychuk
AXXX
Moscow, Russia
andreychuk@cogailab.com

Egor Cherepanov
AXXX & MIRAI
Moscow, Russia
cherepanov.e@miriai.org

Oleg Bulichev
MIRAI & Innopolis University
Moscow, Russia
o.bulichev@innopolis.ru

Alexey Kovalev
AXXX & MIRAI
Moscow, Russia
alexeykkov@gmail.com

Konstantin Yakovlev
SPbU & AXXX
Saint-Petersburg, Russia
k.yakovlev@spbu.ru

Aleksandr Panov
AXXX & MIRAI
Moscow, Russia
panov@axxx.tech

Alexey Skrynnik
AXXX & MIRAI
Moscow, Russia
skrynnikalexey@gmail.com

ABSTRACT

Recent advances in multi-agent reinforcement learning (MARL) have demonstrated success in numerous challenging domains and environments, but typically require specialized models for each task. In this work, we propose a coherent methodology that makes it possible for a single GPT-based model to learn and perform well across diverse MARL environments and tasks, including StarCraft Multi-Agent Challenge, Google Research Football and POGEMA. Our method, MARL-GPT, applies offline reinforcement learning to train at scale on the expert trajectories (400M for SMACv2, 100M for GRF, and 1B for POGEMA) combined with a single transformer-based observation encoder that requires no task-specific tuning. Experiments show that MARL-GPT[1] achieves competitive performance compared to specialized baselines in all tested environments. Thus, our findings suggest that it is, indeed, possible to build a multi-task transformer-based model for a wide variety of (significantly different) multi-agent problems paving the way to the fundamental MARL model (akin to ChatGPT, Llama, Mistral etc. in natural language modeling).

KEYWORDS

Multi-agent Learning; Reinforcement Learning; Multi-Task Learning

ACM Reference Format:
Maria Nesterova, Mikhail Kolosov, Anton Andreychuk, Egor Cherepanov, Oleg Bulichev, Alexey Kovalev, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. 2026. MARL-GPT: Foundation Model for Multi-Agent Reinforcement Learning: AAAI Track. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/BWFP6427

[1]Code: https://github.com/Cognitive-AI-Systems/marl-gpt

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/BWFP6427

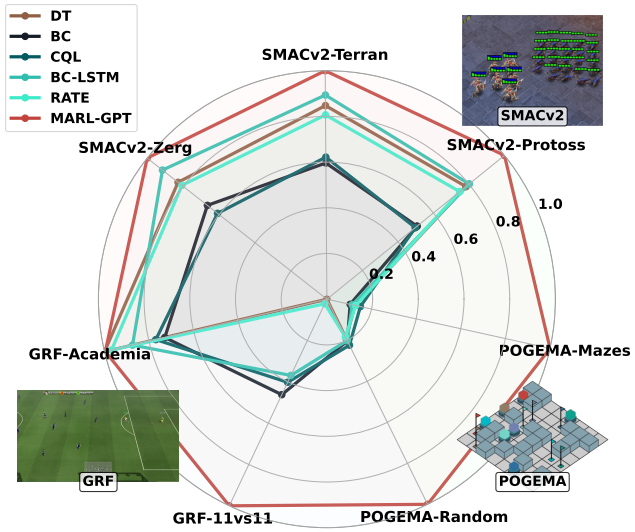

Figure 1: Spider plot demonstrating the relative performance of all evaluated approaches on three different environments – SMACv2, Google Research Football (GRF) and POGEMA.

1 INTRODUCTION

Multi-agent reinforcement learning (MARL) has made significant progress in solving complex tasks such as competitive games like StarCraft or cooperative robotic tasks such as multi-agent pathfinding. However, most MARL methods are designed for a single environment or task, requiring specialized architectures and training pipelines for each new problem. And even in this case the resultant

MARL policies typically struggle when solving the problems not seen in the training time that are different in the number of agents, map layout etc. In this work, we wish not only to improve the generalization abilities of a MARL policy but to make it possible for such policy to efficiently solve tasks from different domains.

To this end we leverage the power of transformer-based architectures (that are the backbone of the overwhelming success in natural language processing tasks and are successfully applied to other domains such as computer vision, robotic control as well) coupled with imitation-learning at scale. We focus on three significantly different MARL environments: Starcraft Multi-Agent Challenge (adversarial combat game) [29], Google Research Football (adversarial sport game) [17], and POGEMA (cooperative multi-robot navigation) [32]. Using the expert policies for those environments that include both learnable policies and the rule-based ones, we obtain a large and diverse dataset of observation-action pairs that are used to train our model, which we call MARL-GPT.

We train MARL-GPT using supervised imitation learning on a diverse dataset of expert demonstrations collected from multiple MARL environments and tasks (Fig. 2). Each expert trajectory consists of sequences of observation-action-reward triplets gathered across scenarios that vary in dynamics, state spaces, and agent interactions. To represent these observations, we tokenize structured, vectorized inputs describing each agent's local view – including itself, allies, and opponents. These tokens are enriched with the learned embeddings that encode the type of feature (e.g., position, health), the agent's identity and team affiliation, and the temporal step of the observation. This flexible encoding supports a variable number of agents and preserves role-specific and group-specific information while maintaining permutation invariance where applicable. The resulting tokens are processed by a transformer encoder, which maps them to expert action predictions and Q-values. Training is performed using standard cross-entropy loss on discrete action outputs or Q-value bins, without relying on recurrence or autoregression. Instead, partial observability is handled by leveraging the transformer's context window and temporal encoding.

As a result we are able to obtain a single (fundamental) MARL model, that generalizes across state and action spaces, reward structures, and agent coordination requirements as confirmed by our extensive empirical evaluation.

Our experiments show that this unified model achieves competitive performance compared to specialized MARL algorithms in all tested domains. This suggests that general-purpose architectures can handle diverse multi-agent tasks without extensive tuning, reducing the need for task-specific designs.

In summary, our key contributions are as follows:

(1) We develop a single GPT-based MARL model that performs notably well across different environments without any architectural changes or fine-tuning.
(2) We create a large dataset of observation-action-reward triplets, vital for imitation learning and offline RL of any MARL policy (not necessarily ours).
(3) We conduct a thorough empirical validation of our model on SMACv2, Google Research Football, and POGEMA, showing competitive results against specialized baselines.

We are committed to open-sourcing the model code, including expert policy weights, training datasets, and final weights of the MARL-GPT model.

2 RELATED WORK

Multi-agent imitation learning (MAIL). In the realm of multi-agent systems, imitation learning and learning from demonstration are widely employed [22, 34]. Imitation learning in multi-agent scenarios, also known as MAIL, is a problem in which agents learn to perform a task in a multi-agent system by observing and mimicking expert demonstrations without any knowledge of the reward function from the environment. This approach has gained particular traction in the context of controlling urban traffic and traffic lights at intersections [2, 13], controlling the power in wireless networks [44] due to the availability of a vast amount of data collected in real-world scenarios and the use of high-quality simulators, e.g. Sumo [23] in traffic applications.

Within the realm of MAIL, there are various methods to consider, including those that employ Bayesian approaches [41], generative adversarial techniques [20, 33], statistical tools for capturing interdependencies between agents [37], low-rank subspaces [30], latent models for coordinating agents [18], decision transformers [24, 39], and more. Demonstrations are frequently used for pre-training in games, such as learnable models for chess [28, 31], and in multi-agent pathfinding tasks, as exemplified by MAPF-GPT [1] and SCRIMP [38]. Despite the presence of models that are trained on pre-collected data, they are often specialized for a specific environment or even a single task within that environment. These models have limited generalizability even to conditions within the same environment, and their performance is significantly reduced when learning in a multitasking setting.

Foundation models for multi-agent systems. Foundation models are typically trained on large datasets, enabling zero-shot or few-shot learning [3, 42]. From the perspective of an autonomous agent, it is a model that can perform new tasks that are different from those it was trained on, either with additional demonstration of desired behavior or without any demonstration [9]. Another important feature of these foundation models is their fine-tuning ability for specific tasks, allowing them to improve performance quickly (see Gato tuning [27]). These models are commonly used in robotics to solve multimodal tasks where the goal is specified in text instructions [9, 15, 35]. However, in multi-agent scenarios, such models are less common, with examples including the Magnetic-One model for language and multimodal task-solving in WebArena [10], as well as the MAPF-GPT model for pathfinding [1].

Work on multitasking multi-agent models is also close to this topic, in which attempts are made to train a single policy for action in different scenarios and environments [26]. The primary direction for developing foundation models in the multi-agent setting is based on transformer architectures. First, transformers can process observations of varying lengths [12], which is particularly important when the dimensionality of the observation depends on the number of agents in the task. Second, transformers enable the model to capture inter-agent relationships, which is valuable not only for observation processing, but also for partially centralized training

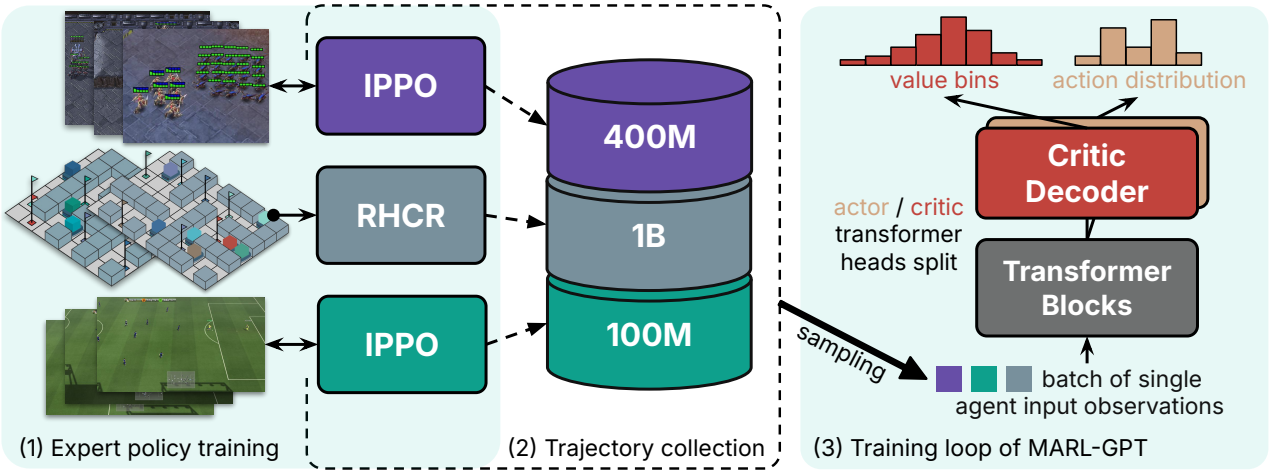

Figure 2: The general pipeline begins with training expert policies across diverse MARL environments using domain-appropriate algorithms (e.g., IPPO, RHCR). These policies generate large-scale datasets of observation-action(-reward) trajectories. MARL-GPT is then trained on the aggregated data using cross-entropy loss to predict expert actions and Q-values, enabling a single model to generalize across environments, tasks, and coordination regimes.

approaches [39]. Additionally, several offline methods [21, 43] extract coordination skills that are generalized between grouping agents. However, these approaches lack the key properties of foundation models, particularly the ability to fine-tune on additional demonstrations, and they struggle to effectively train in multi-task settings with distinct environments.

Some papers have explored the possibility of adapting single-agent foundation models to solve multi-agent problems without modifying the pre-training process [36, 40]. In our method, we focus on the task of training a foundation model for multiple multi-agent environments from scratch. It should be noted that no such single pre-trained model has been proposed yet. This is due to the complex nature of multi-agent policies in various tasks, such as StarCraft [29] and football [17], and the lack of large expert trajectory datasets necessary for effective foundation model training. In our MARL-GPT model, we overcome these difficulties and show that it is possible to train a multitasking, multi-agent foundation model on a reasonable set of expert data and with expert-level quality.

3 BACKGROUND

Multi-Agent Reinforcement Learning extends single-agent reinforcement learning (RL) to domains with multiple interacting decision makers. In partially observable environments, this interaction can be formalized by a decentralized partially observable Markov decision process (Dec-POMDP) [25]. This is a tuple $(N, \mathcal{S}, \mathcal{A}, T, \{O^i\}_{i\in N}, O, \{\mathcal{R}^i\}_{i\in N}, \gamma)$, where N is the agent set, $\mathcal{S}$ is the global state space, and all agents share a common action space $\mathcal{A}$. The joint action is $\mathbf{a} = (a^1, \ldots, a^n) \in \mathcal{A}^n$. The transition kernel $T(s' \mid s, \mathbf{a})$ specifies the probability of moving from state s to s' given the joint action. Each agent receives a private observation $o^i \in O^i$ drawn from the observation function $O(o^1, \ldots, o^n \mid s, \mathbf{a}, s')$. Rewards may differ across agents, with $\mathcal{R}^i(s, \mathbf{a}, s')$ denoting the signal for agent i, and $\gamma \in [0, 1)$ the discount factor.

At each time step t, agent i selects its action based on its own interaction history via a policy $\pi^i : (o_0^i, a_0^i, \ldots, o_t^i) \mapsto \Delta(\mathcal{A})$, where $\Delta(\cdot)$ denotes the distribution of actions provided by the policy. After all agents act, the environment transitions and each agent receives $r_t^i = \mathcal{R}^i(s_t, \mathbf{a}_t, s_{t+1})$. The learning objective in MARL is for each agent i to find an optimal policy $\pi^{*\,i}$ that maximizes its expected cumulative return, which is a discounted sum of step-wise rewards.

Each agent deploys its learned policy, which selects actions based only on its own local observation history. The policies themselves are conditioned only on local information, making execution fully decentralized. Agents do not need to communicate or observe others' states/actions at runtime.

Parameter sharing employs a single policy network π across all agents. This approach reduces the parameter count and enables efficient knowledge transfer, though it may limit policy diversity when agents require specialized behaviors.

Offline RL and imitation learning methods train on expert trajectories $\mathcal{D}$. The simplest approach is behavior cloning [14], which formulates imitation learning as supervised learning by minimizing the cross-entropy loss $\mathbb{E}_{o_t, a_t \sim \mathcal{D}} \left[\mathcal{L}_{CE}(a \sim \pi(a|o_t), a_t)\right]$ over expert demonstrations (o_t, a_t).

4 METHOD

The suggested pipeline of obtaining a multi-task MARL model can be divided into three main components. First, expert policies for each task are acquired, and expert trajectories are collected using them. Second, observations are encoded in a way that enables the model to operate across multiple environments, in contrast to the expert policies, which are specialized for a single environment. Finally, a transformer architecture is trained via behavior cloning to imitate the expert behavior.

4.1 Acquiring of Expert Policies and Data Collection

Let π_e^* denote an expert policy for the environment $e \in \mathcal{E}$. These expert policies serve as sources of high-quality demonstrations and can be obtained through a variety of means. In certain well-established domains, such as chess, expert policies are readily available in the form of rule-based or search-based engines (e.g., Stockfish with NNUE), which do not require further training. These systems can generate large volumes of expert trajectories with negligible additional cost.

In contrast, expert policies for complex multi-agent systems are significantly less common. One exception is the domain of multi-agent pathfinding, where centralized solvers (e.g., Conflict-Based Search) can be used to compute globally optimal or near-optimal solutions. These solvers are environment-specific but provide valuable expert supervision when applicable.

In many modern MARL environments, no public expert policies are readily available. In such cases, we construct strong policies by applying large-scale reinforcement learning. For instance, we use independent PPO agents trained separately in each environment e to convergence, resulting in a set of high-performing task-specific policies. This approach is applicable even in partially observable settings.

Each expert policy π_e^* is used to collect trajectories by interacting with its corresponding environment. If the policy is recurrent, it may condition on a history of past observations. Specifically, the expert produces a trajectory:

$$\tau_e = \{(o_t, a_t, r_t)\}_{t=1}^{T},$$

where o_t is the observation at time t, and $a_t \sim \pi_e^*(\cdot \mid h_t)$ is the action sampled based on observation history $h_t = \{o_{t-k}, \ldots, o_t\}$. The agent receives the reward r_t. In practice, the policy may use a GRU or similar recurrent architecture to encode h_t into a compact hidden state.

We collect expert trajectories across a distribution of environments $\mathcal{E}$. Some of them may contain multiple different scenarios (tasks), allowing us to gather data that vary not only across environments, but also across tasks within a given environment. This results in a diverse dataset of expert demonstrations covering a wide range of state spaces, dynamics, and task structures.

4.2 Observation Encoding

We consider the task of multi-agent decision making in a partially observable environment. In this setting, observations usually include information about the agent itself, the surrounding agents, and the environment around them. We assume that observation from any environment can be represented as a vector $o = \{o_i\}_{i=0}^{n}$. For each element o_i in this observation vector, we obtain a token embedding tok_i by applying a learned linear layer $tok_i = L(o_i)$ that projects the raw input into the embedding space. L is the same for all environments and maps a number into a vector.

Each element of the observation corresponds to a specific environmental property, e.g. the health of the nearest agent. To provide context to these elements, we augment them with positional and structural information. Four positional numbers are defined to encode this information. First, pos_{attr} (attribute) indicates the type of observed property. For instance, whether it is the agent's health or its coordinates. If both agent k and agent j have the same property P observed, they will share the same pos_{attr}. Second, pos_{team} denotes the group to which the observed agent belongs. Agents may be divided into different groups based on their roles or goals, such as allies and enemies. Third, agents can be numbered within their group using pos_{indx} . Thus, an observation element o_i can be associated with an agent of the group pos_{team} and the index pos_{indx}. The agent whose observation is being considered is always assigned group 0 and index 0. All global properties of the situation are also attributed to this agent (group 0, index 0).

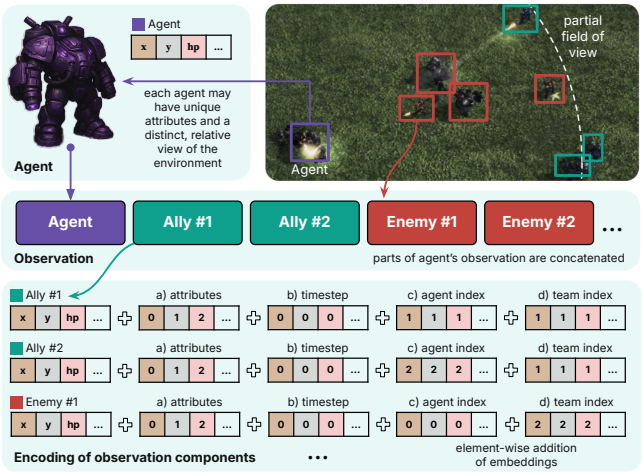

Figure 3: Illustration of the proposed encoding scheme for multi-agent systems, demonstrated using an example from the SMACv2 environment. Each agent receives a structured, vectorized observation containing information about itself, its allies, and nearby enemies. For each observed agent, the input features (such as position, health, etc.) are enriched with additional embeddings: (a) a positional encoding over the feature dimensions, (b) a global timestep encoding, (c) an agent identity index, and (d) a team index distinguishing allies from enemies. These components are combined via element-wise addition, resulting in a contextualized embedding for each observed agent that can be processed by a transformer encoder. The provided encoding scheme is general and transferable across multi-agent systems. By augmenting raw observation features with agent- and team-specific indices, the encoding enables the model to distinguish between different functional groups (e.g., allies vs. enemies) and individual agents, regardless of environment-specific dynamics or layouts. This structure preserves permutation invariance where appropriate, while still allowing the model to learn role-specific interactions. The encoding supports a dynamic number of agents, limited only by the model's maximum context size.

The environment is partially observable, which means that the agent does not have access to the full state of the environment. One simple way to handle this limitation is to include in the final observation a history of the last h observations from the environment, or

to complete the current observation with some information from previous steps. To indicate the time moment of each observation element, we add a positional encoding pos_{time}. It is important to note that we do not use methods such as autoregressive modeling or recurrent neural networks in this approach.

Thus, each observation element o_i can be represented by a tuple of four components: agent index pos^i_{indx}, group number pos^i_{team}, agent property number pos^i_{attr}, and time step pos^i_{time} from which the observation was taken. The structure of these positional vectors depends on the specific environment. For each positional component, a corresponding embedding is learned $emb^i_{type} = L_{type}(pos^i_{type})$ and added to the respective token.

In summary, each observation element o_i is contextualized by a four-component positional vector: the agent index pos^i_{indx}, group identifier pos^i_{team}, attribute type pos^i_{attr}, and timestep pos^i_{time}. While the rules for assigning these positional indices are environment-dependent, the mechanism for using them is universal. Each positional component is mapped to a vector using a learned embedding layer $emb^i_{type} = L_{type}(pos^i_{type})$. These four positional embeddings are then summed with the initial observation token tok_i to produce the final input representation, res_i.

$$res_i = tok_i + emb^i_{indx} + emb^i_{team} + emb^i_{attr} + emb^i_{time}.$$

The result tokens res_i are then processed by a GPT-like model. The example of a positional encoding for the well-known SMACv2 environment is shown in Fig. 3. Appendix A provides recommendations for constructing positional encodings tailored to specific environment.

4.3 Model Training

We study a universal model that predicts an agent's action using only its own observation, without access to the state or observations of other agents. Our approach is based on offline RL with a data-driven actor-critic architecture. At each timestep t, the transformer-based observation encoder processes the agent's observation o_t into a hidden state h_t, which is shared by both the critic Q and stochastic policy π output heads. The model is trained by minimizing joint loss $\mathcal{L} = \mathcal{L}_{critic} + \mathcal{L}_{actor}$ using minibatch stochastic gradient descent with the Adam optimizer.

Critic. Classification with categorical cross-entropy loss ($\mathcal{L}_{CE}$) is more effective than regression with mean squared error when training large neural networks, especially transformer architectures. For this reason, we use a discrete critic approach [8, 28]. The Q-value range is divided into a fixed number of uniform bins, and each Q-value is represented as a probability distribution over these bins. If K is the scalar, then K^B represents this scalar as a vector of bins.

The critic loss consists of two parts: the temporal difference (TD) error and the conservative regularization (CR). The TD target is $y_t = r_t + \gamma\ \bar{Q}(o_{t+1}, a \sim \pi(a|o_t))$, where $\bar{Q}$ and $\bar{\pi}$ refer to frozen target networks. The regularization term addresses the issue of overestimating Q-values for observation-action pairs that are rarely or never seen in the dataset [4, 16]. To prevent this, we penalize the Q-values for actions with low probability under the current policy, pushing them toward a minimal attainable value $q_{\min}$. Specifically, we define $\pi'(a \mid o) = \frac{1-\pi(a|o)}{Z}$ as a distribution over actions that have a low probability according to π, where Z is a normalization constant. The critic loss is:

$$\mathcal{L}_{critic} = \alpha_{td}\ \mathbb{E}_{o,a\sim\mathcal{D}}\left[\mathcal{L}_{CE}(Q^B(o,a), y^B_t)\right] + \\ + \alpha_{cr}\ \mathbb{E}_{o\sim\mathcal{D},\ a\sim\pi'(a|o)}\left[\mathcal{L}_{CE}(Q^B(o,a), q^B_{min})\right]. \quad (1)$$

Policy. The policy loss combines behavior cloning (BC) loss and actor loss (A). We calculate actor loss using the advantage function $A(o_t, a_t)$ because it offers a stable and scale-invariant learning signal. This signal helps the actor focus on actions that perform better than the average, rather than just their absolute value.

$$A(o_t, a_t) = Q(o_t, a_t) - \mathbb{E}_{a\sim\pi(a|o_t)}\left[Q(o_t, a)\right]$$

We include a behavior cloning loss in the policy training because it guides the model to imitate the best behavior observed in the dataset. This loss provides a strong and reliable supervisory signal, especially when the optimal policy is unknown or difficult to estimate. The policy loss is:

$$\mathcal{L}_{actor} = -\alpha_a\ \mathbb{E}_{o_t,a_t\sim\mathcal{D}}\left[A(o_t, a_t)\log\pi(a_t|o_t)\right] \\ + \alpha_{bc}\ \mathbb{E}_{o_t,a_t\sim\mathcal{D},\ a\sim\pi(a|o_t)}\left[\mathcal{L}_{CE}(a, a_t)\right]. \quad (2)$$

Action Space. To handle varying action spaces across environments, a universal output layer (one transformer layer plus a linear layer) is used, combined with environment-specific action masking. This masking ensures that the probability of unavailable actions is always zero, allowing the agent to adapt to different scenarios while maintaining a consistent architecture.

Unified model for all environments. The model architecture and all training hyperparameters are identical across environments. Only the positional encoding and the action mask are environment dependent.

Online fine-tune. During offline training, the agent learns mainly from successful trajectories but tends to underestimate Q-values for unseen observations and actions. This leads to incorrect Q-value estimates during the online phase. To mitigate this, the actor is frozen for the initial n steps of online training while the critic is pretrained using online interactions with the environment. Afterward, online fine-tuning is performed using PPO, which helps stabilize learning and adapt the policy safely to new data.

5 EMPIRICAL EVALUATION

5.1 Experimental Setup

5.1.1 Environments. To evaluate MARL-GPT, we selected three well-known multi-agent environments. SMACv2 [7] is a challenging real-time strategy benchmark that tests coordinated decision-making and teamwork in complex combat. GRF (Google Research Football) [17] provides a dynamic, stochastic setting with varied strategies, assessing policy adaptability and temporal reasoning. The POGEMA benchmark[2] [32] involves multi-agent pathfinding in grid worlds, providing a tough setting for testing scalable generalization to new agent populations and scenarios. We use the lifelong scenario, where a new goal is generated when an agent completes previous one.

[2]https://github.com/Cognitive-AI-Systems/pogema-benchmark

		Single-Environment Baselines					Multi-Env	Single-Task
Environment	**Task**	DT	BC	CQL	BC-LSTM	RATE	MARL-GPT	Expert
SMACv2	protoss 5_vs_5	82 ± 3	61 ± 3	57 ± 3	85 ± 3	79 ± 3	89 ± 3	87 ± 3
	protoss 5_vs_6	30 ± 4	12 ± 4	14 ± 4	30 ± 4	28 ± 4	54 ± 4	49 ± 4
	terran 5_vs_5	84 ± 3	69 ± 3	69 ± 3	88 ± 3	85 ± 3	93 ± 2	91 ± 2
	terran 5_vs_6	48 ± 4	24 ± 4	28 ± 4	51 ± 4	41 ± 4	63 ± 4	61 ± 4
	zerg 5_vs_5	65 ± 3	56 ± 3	50 ± 3	72 ± 3	64 ± 3	74 ± 3	72 ± 3
	zerg 5_vs_6	34 ± 4	24 ± 4	23 ± 4	38 ± 4	33 ± 4	46 ± 4	48 ± 4
GRF	pass and shoot	80 ± 2	44 ± 2	60 ± 2	90 ± 2	78 ± 2	96 ± 2	97 ± 2
	corner	60 ± 5	37 ± 4	30 ± 4	22 ± 4	58 ± 5	43 ± 5	40 ± 4
	counterattack easy	88 ± 3	86 ± 3	87 ± 3	88 ± 3	85 ± 3	89 ± 3	89 ± 3
	11 vs 11 easy	0 ± 0	40 ± 3	38 ± 3	43 ± 4	4 ± 3	98 ± 2	99 ± 1
	11 vs 11 medium	0 ± 0	41 ± 3	35 ± 4	30 ± 3	1 ± 1	98 ± 3	100 ± 0
	11 vs 11 hard	0 ± 0	40 ± 3	34 ± 4	24 ± 5	1 ± 1	68 ± 7	94 ± 1
POGEMA	Random	0.22 ± 0.01	0.24 ± 0.01	0.26 ± 0.01	0.23 ± 0.01	0.21 ± 0.01	1.16 ± 0.04	2.16 ± 0.13
	Mazes	0.12 ± 0.01	0.10 ± 0.01	0.14 ± 0.01	0.11 ± 0.01	0.12 ± 0.01	0.96 ± 0.04	1.55 ± 0.08
	Warehouse	0.13 ± 0.01	0.14 ± 0.01	0.17 ± 0.01	0.12 ± 0.01	0.13 ± 0.01	1.02 ± 0.01	
	Cities-tiles	0.68 ± 0.03	0.46 ± 0.02	0.66 ± 0.03	0.49 ± 0.02	0.65 ± 0.03	2.72 ± 0.12	

Table 1: Win rates % (for SMACv2 and GRF) and average throughput (for POGEMA). Higher is better. The table compares MARL-GPT (Multi-Domain) to expert single-task policies and standard offline RL baselines (Single-Domain) across SMACv2, GRF, and POGEMA. MARL-GPT shows strong generalization, often matching or outperforming the expert and baselines. The highlighting indicates the best-performing model (excluding single-task experts).

5.1.2 Expert Policy Generation. To prepare the expert dataset for SMACv2 and GRF, we used large-scale asynchronous IPPO[3]. In each SMACv2 training environment, IPPO was trained using a total of 1 billion collected observations across scenarios involving *Protoss*, *Terran*, and *Zerg*, with agent populations in both balanced (5vs5, 10vs10) and imbalanced (5vs6) settings. The resulting policy was then used to obtain 400 million agent training samples. We trained expert policies for each GRF scenario using IPPO. For the full 11vs11 match, we trained three distinct expert policies, one for each opponent difficulty level: easy, medium, and hard. Each of these policies was trained for 20 billion environment steps. For the three standard Academy scenarios, *Pass and Shoot with Keeper* ran for 300 million timesteps, *Easy Counter-attack* for 400 million timesteps, and *Corner* for 200 million timesteps. For the POGEMA environment, we used a centralized solver called RHCR [19], in contrast to IPPO, which serves as the expert policy in other settings. RHCR operates by decomposing lifelong MAPF into windowed planning instances, resolving collisions within bounded time horizons while having access to the full state of the environment. The performance of such a centralized policy serves as an upper bound for any learnable policy, which does not have access to the full state.

5.1.3 Trajectory Collection. We then used these pre-trained experts to generate multi-agent trajectories. At every timestep in each scenario, we recorded the observations o_t (as floats), the expert actions a_t, available actions m_t, scalar rewards r_t, and done flags d_t for each agent. The tuple $(o_t, a_t, m_t, r_t, d_t)$ represents a single element in the dataset. For each SMACv2 task, we collected 60 million data elements for tasks with 5 agents and 80 million for those with 10 agents. For GRF this produced approximately 200,000 trajectories from *Pass and Shoot with Keeper*, 25,000 from *Easy Counter-attack*, 75,000 from *Corner*, and 30,000 from the full 11vs11 match. For POGEMA, we collected 1 billion elements in mazes and 120 million on random maps, using 32 agents and 256 steps per episode. We designed a custom dense reward for POGEMA: agents receive 1 when moving toward the goal, and 0 otherwise.

5.1.4 Training. All versions of MARL-GPT were trained using 7M parameters, a history window of 6 (except POGEMA), and the full dataset (except SMACv2 tasks with 10 agents). The main model was trained for 500,000 timesteps, while models used for testing theories were trained for 10,000. Changes in parameters are explicitly noted. During training, each batch was balanced to provide equal data from all environments, ensuring that the model receives the same proportion of samples per environment. The loss-weighting hyperparameters were fixed and shared across all environments without tuning. The components were chosen to ensure that the actor and critic losses are on a comparable scale, promoting balanced gradient flow.

5.1.5 Model evaluation. We assess the model's performance on all tasks within the training distribution for the SMACv2 and GRF domains by running 500 evaluation episodes per task and computing the average win rate. For the pathfinding problem, evaluation is conducted using the POGEMA benchmark on different map sets: random maps and mazes for training tasks, and Warehouse and Cities-tiles for unseen tasks. Performance in POGEMA is measured using the average throughput metric, defined as the ratio of the total number of goals achieved by all agents to the episode length.

[3]https://github.com/alex-petrenko/sample-factory

5.2 Experimental Results

5.2.1 Multi-task MARL. Table 1 shows the complete experimental results across all three environments. In these experiments, we compare MARL-GPT (trained jointly on all environments) to five standard offline RL baselines (each trained on all tasks within a single environment). The evaluated baselines include two **memory-free** methods: Behavior Cloning (BC) and Conservative Q-Learning (CQL; Kumar et al. [16]); three **memory-based** methods: Decision Transformer (DT; Chen et al. [5]), Recurrent Action Transformer with Memory (RATE; Cherepanov et al. [6]) and a recurrent BC variant with a Long Short-Term Memory [11] backbone (BC-LSTM). Crucially, all offline RL baselines are trained on the same raw observation vectors without the positional encodings: attribute, agent-index, team-index, and timestep that MARL-GPT injects into every token (see 3).

5.2.2 Expert quality. For SMACv2 and GRF, the expert policy is a learned model that is not near-optimal. In contrast, for POGEMA, the expert is a centralized planner (RHCR) that leverages full environment observability and a heavy search to resolve conflicts, providing strong demonstrations that serve as a useful upper bound. We aim to study the components of our model in settings where the expert offers such high-quality supervision.

Table 2 presents results for RHCR, MAPF-GPT (a transformer model trained with behavior cloning on MAPF tasks), MARL-GPT, and MARL-GPT-BC (behavior cloning only). In this setting, behavior cloning variants outperform MARL-GPT. This highlights that MARL-GPT's reward-based loss components may introduce instability when learning from strong expert data, and that designing effective reward functions in multi-agent pathfinding remains challenging due to the need to capture coordination, conflict resolution, and long-term planning. As a result, imitation-based approaches can more effectively leverage high-quality demonstrations like those provided by RHCR.

Scenario	MARL-GPT	MARL-GPT-BC	MAPF-GPT	RHCR
Random	1.16	1.46	1.50	2.16
Mazes	0.96	1.00	1.09	1.55
Warehouse	1.02	1.47	1.27	2.35
Cities-tiles	2.72	2.71	2.99	3.48

Table 2: Average throughput. Comparing MARL-GPT with behavior cloning variants (MARL-GPT-BC, MAPF).

5.2.3 Unseen tasks. For the SMACv2 environment, we evaluated the model on previously unseen tasks with 10vs10 and 7vs7 agents, which are challenging for models trained only on 5 agents. To address this, we collected additional datasets with 10 agents and conducted two experiments: (1) We fine-tuned the pre-trained model on a small (9M) dataset with 10 agents across all races for 2,000 training steps; (2) we trained a new model from scratch using all 5-agent tasks plus an additional terran_10_vs_10 task (for 3,000 and 10,000 training steps).

The results (Table 3) show that the model can generalize to new maps with more agents if the training data include tasks with similar agent counts. Moreover, fine-tuning with dataset the pre-trained model requires less data and fewer steps to adapt to a new map than training from scratch. Two experiments show better generalization to unseen tasks. For example, (3) a model trained without terran data still performs well on new terran 5vs5 and 5vs6 scenarios. Additionally, training the model without BC loss across all races (4) can improve performance on tasks with more agents. However, including BC loss helps the model achieve better results on the training tasks but reduces its ability to generalize to new scenarios. All experiments were conducted with history length 4.

Method / Environment	From scratch (2)		Zero-shot			Fine-tune
	3k	10k	MARL-GPT	w/o BC (4)	w/o terran (3)	2k (1)
terran 5_vs_5	87	91	88	80	44*	92
terran 5_vs_6	48	57	52	41	16*	62
terran 10_vs_10	71	83	0*	47*	0*	88
protoss 10_vs_10	62*	61*	0*	8*	0*	81
zerg 10_vs_10	29*	22*	1*	28*	0*	43
terran 7_vs_7	80*	86*	16*	60*	2*	88*

Table 3: Win rates %. Comparing variants of methods on unseen tasks: models trained from scratch and pretrained (zero-shot and fine-tuned). Each experiment used different datasets but always included tasks with zerg and protoss races (5vs5 and 5vs6). Tasks excluded from training and tested zero-shot are highlighted and marked with * next to their success rate. The best zero-shot results are highlighted with a stronger emphasis.

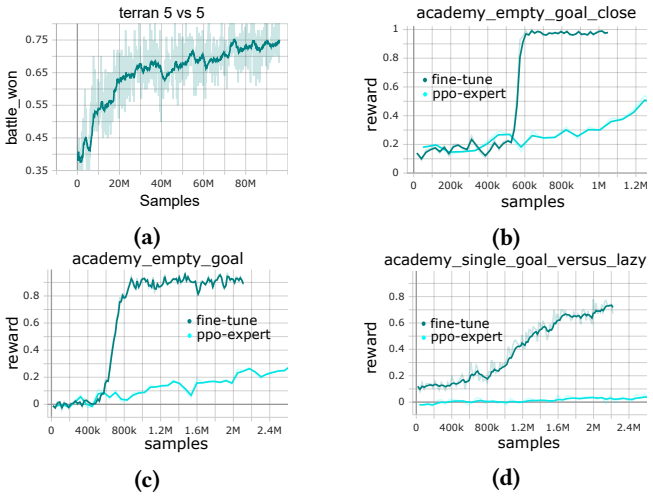

Figure 4: Online fine-tuning results: (a) Battle won on the terran 5vs5 task (SMACv2). (b–d) Comparison of fine-tuned MARL-GPT and an expert model trained from scratch on GRF.

5.2.4 Online fine-tune. Online fine-tune algorithm was tested on unseen tasks, in which MARL-GPT in zero-shot format achieved non-zero results. In SMACv2, we tested online fine-tuning on a new race: the initial weights were trained without the terran dataset (w/o terran in Table 3). Pretraining of the critic used 2M samples. The target task was terran 5_vs_5. At the beginning, the success rate (SR) was 0.4, and it improved to 0.8 by the end (Fig. 4a). For GRF, we tested on several unseen tasks (academy_empty_goal_close, academy_empty_goal and academy_single _goal_versus_lazy) comparing results with an expert trained from scratch. Pretraining of the

critic used 500k samples. Fine-tuning the pretrained model on the GRF environment showed faster adaptation to new tasks compared to training a small model from scratch (Fig. 4).

5.2.5 Ablation study. We investigated how different parameters influence the final model's performance (Table 4). All models were trained under identical conditions, except for one varied parameter. MARL-GPT, our final model, uses 7M parameters, a long history window of 6, positional encoding, and the full dataset. Our results show that model size, history length (notably for GRF), training data size, and positional encoding have a substantial impact on performance.

	No	Model Size		Dataset		History Length		γ
Task	Pos. Enc.	2M	7M	Half	Small	2	4	0.99
terran 5_vs_6	41	37	53	52	56	49	50	39
zerg 5_vs_6	33	36	40	38	40	37	39	29
corner	29	30	30	32	42	0	32	29
counterattack	88	85	88	87	20	20	0	85
maze	0.56	0.38	0.75	0.70	0.68	0.63	0.66	0.22

Table 4: Ablation study across tasks from SMACv2, GRF, and POGEMA. Reported are win rates (SMACv2 and GRF) and average throughput (POGEMA). We compare MARL-GPT (7M parameters) against a smaller 2M variant, and evaluate the impact of removing positional encoding, reducing dataset size, and varying history length.

5.2.6 Real-World Mini Experiment. We conducted a real-robot MAPF experiment; see Appendix D for details.

5.3 Implementation Details

The open-source repository[4] provides all resources required to reproduce the MARL-GPT experiments, including scripts for offline training, online fine-tuning, and evaluation.

The offline training MARL-GPT code is based on NanoGPT[5] framework. This framework was selected for its straightforward and modular design, allowing for easy modification and adaptation. Training the 7M-parameter model in the main experiment for 500K iterations took 161 hours using 8 NVIDIA H100 80GB GPUs. In other experiments, training the same model for 10K iterations took 13 hours on 2 NVIDIA H100 80GB GPUs. Table 5 provides the list of hyperparameters used in our main experiments. When the model was fine-tuned with 10-agent SMACv2 data (Chapter 5.2.3) mixed with batches of original tasks to prevent catastrophic forgetting, no performance drop occurred.

6 CONCLUSION

We presented MARL-GPT, a unified transformer-based model for multi-agent reinforcement learning that operates across diverse environments using a single architecture. Trained purely from expert trajectories via imitation learning and RL, MARL-GPT achieves competitive or superior performance compared to specialized baselines in SMACv2, GRF, and POGEMA. Our results demonstrate the viability of a generalist MARL model, suggesting a path forward toward scalable, foundation models for multi-agent decision-making.

[4]MARL-GPT: https://github.com/Cognitive-AI-Systems/marl-gpt
[5]NanoGPT: https://github.com/karpathy/nanoGPT

Parameter	Value
Total training iterations (main experiment)	500,000
Total training iterations (small experiment)	10,000
Batch size	900
Max size of observation	700
Action size	20
Number critic bins for each action	20
Max value for critic regularization	5
α_{td}	1
α_{cr}	1
α_a	0.1
α_{bc}	1
history length (main experiment)	6
history length (small experiments)	4
Value size per one action	20
γ	0.95
Target Network Update	0.7
Minimum learning rate	6e-5
Maximum learning rate	6e-4
Learning rate decay	cosine
AdamW optimizer beta1	0.9
AdamW optimizer beta2	0.95
Gradient clipping	1.0
Weight decay	1e-1
Warm-up iterations	2000
Use PyTorch 2.0 compilation	True
Gradient accumulation steps	16
Number of layers	8
Number of attention heads	8
Embedding size	256

Table 5: Learning Hyperparameter Details.

REFERENCES

[1] Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. 2025. Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 23126–23134.

[2] Raunak P Bhattacharyya, Derek J Phillips, Blake Wulfe, Jeremy Morton, Alex Kuefler, and Mykel J Kochenderfer. 2018. Multi-agent imitation learning for driving simulation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1534–1539.

[3] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[4] Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. 2023. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*. PMLR, 3909–3928.

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34 (2021), 15084–15097.

[6] Egor Cherepanov, Aleksei Staroverov, Alexey Kovalev, and Aleksandr Panov. 2026. Recurrent Action Transformer with Memory. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=kByN4v0M3e

[7] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. 2023. SMACv2: An Improved Benchmark for Cooperative Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 37567–37593. https://proceedings.neurips.cc/paper_files/paper/2023/file/764c18ad230f9e7bf6a77ffc2312c55e-Paper-Datasets_and_Benchmarks.pdf

[8] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. 2024. Stop regressing: training value functions via classification for scalable deep RL. In *Proceedings of the 41st International Conference on Machine Learning*. 13049–13071.

[9] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. 2023. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research* (2023).

[10] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. 2024. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468* (2024).

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[12] Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. 2021. {UPD}eT: Universal Multi-agent {RL} via Policy Decoupling with Transformers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=v9c7hr9ADKx

[13] Chang Huang, Junqiao Zhao, Hongtu Zhou, Hai Zhang, Xiao Zhang, and Chen Ye. 2023. Multi-agent Decision-making at Unsignalized Intersections with Reinforcement Learning from Demonstrations. In *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1–6.

[14] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 1–35.

[15] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2024. OpenVLA: An Open-Source Vision-Language-Action Model. In *8th Annual Conference on Robot Learning*.

[16] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.

[17] Karol Kurach, Anton Raichuk, Piotr Stanczyk, Michal Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. 2020. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 4501–4510.

[18] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. 2017. Coordinated multi-agent imitation learning. In *International Conference on Machine Learning*. PMLR, 1995–2003.

[19] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar, and Sven Koenig. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*. 11272–11281.

[20] Wei Li, Shiyi Huang, Ziming Qiu, and Aiguo Song. 2024. GAILPG: Multi-Agent Policy Gradient with Generative Adversarial Imitation Learning. *IEEE Transactions on Games* (2024).

[21] Sicong Liu, Yang Shu, Chenjuan Guo, and Bin Yang. 2025. Learning Generalizable Skills from Offline Multi-Task Data for Multi-Agent Cooperation. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=HR1ujVR0ig

[22] Shicheng Liu and Minghui Zhu. 2024. Learning multi-agent behaviors from distributed and streaming demonstrations. *Advances in Neural Information Processing Systems* 36 (2024).

[23] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2575–2582.

[24] Linghui Meng, Muning Wen, Chenyang Le, Xiyun Li, Dengpeng Xing, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, Yaodong Yang, et al. 2023. Offline pre-trained multi-agent decision transformer. *Machine Intelligence Research* 20, 2 (2023), 233–248.

[25] Frans A Oliehoek, Christopher Amato, et al. 2016. *A concise introduction to decentralized POMDPs*. Vol. 1. Springer.

[26] Hyunwoo Park, Baekryun Seong, and Sang-Ki Ko. 2025. SPECTra: Scalable Multi-Agent Reinforcement Learning with Permutation-Free Networks. *arXiv preprint arXiv:2503.11726* (2025).

[27] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A Generalist Agent. *Transactions on Machine Learning Research* (2022). https://openreview.net/forum?id=1ikK0kHjvj

[28] Anian Ruoss, Gregoire Deletang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, Cannada A Lewis, Joel Veness, and Tim Genewein. 2024. Amortized Planning with Large-Scale Transformers: A Case Study on Chess. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[29] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2186–2188.

[30] Andy Shih, Stefano Ermon, and Dorsa Sadigh. 2022. Conditional imitation learning for multi-agent games. In *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 166–175.

[31] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[32] Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. 2025. POGEMA: A Benchmark Platform for Cooperative Multi-Agent Pathfinding. In *The Thirteenth International Conference on Learning Representations*.

[33] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. 2018. Multi-agent generative adversarial imitation learning. *Advances in neural information processing systems* 31 (2018).

[34] Jingwu Tang, Gokul Swamy, Fei Fang, and Steven Wu. 2024. Multi-Agent Imitation Learning: Value is Easy, Regret is Hard. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[35] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. 2024. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213* (2024).

[36] Rishi Veerapaneni, Arthur Jakobsson, Kevin Ren, Samuel Kim, Jiaoyang Li, and Maxim Likhachev. 2024. Work Smarter Not Harder: Simple Imitation Learning with CS-PIBT Outperforms Large Scale Imitation Learning for MAPF. *arXiv preprint arXiv:2409.14491* (2024).

[37] Hongwei Wang, Lantao Yu, Zhangjie Cao, and Stefano Ermon. 2021. Multi-agent imitation learning with copulas. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part I 21*. Springer, 139–156.

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

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

[40] Liming Xu, Sara Almahri, Stephen Mak, and Alexandra Brintrup. 2024. Multi-agent systems and foundation models enable autonomous supply chains: Opportunities and challenges. *IFAC-PapersOnLine* 58, 19 (2024), 795–800.

[41] Fan Yang, Alina Vereshchaka, Changyou Chen, and Wen Dong. 2020. Bayesian multi-type mean field multi-agent imitation learning. *Advances in Neural Information Processing Systems* 33 (2020), 2469–2478.

[42] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. 2023. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129* (2023).

[43] Fuxiang Zhang, Chengxing Jia, Yi-Chen Li, Lei Yuan, Yang Yu, and Zongzhang Zhang. 2022. Discovering generalizable multi-agent coordination skills from multi-task offline data. In *The Eleventh International Conference on Learning Representations*.

[44] Yiming Zhang, Kun Yang, Cong Shen, and Dongning Guo. 2025. Multi-Agent Decision Transformer for Power Control in Wireless Networks. In *2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1–5.