Encoding Goals as Graphs: Structured Objectives for Scalable Cooperative Multi-Agent Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Many cooperative multi-agent tasks are naturally defined by graph-structured objectives, where agents must collectively reach, for example, a desired relational configuration or satisfy a set of constraints. These objectives often encode spatial arrangements, inter-agent relations, or constraints that can be formalized as target graphs. However, current goal-conditioned multi-agent reinforcement learning (MARL) algorithms do not employ these symbolic and structured representations to direct their agents towards effective strategies. We propose Graph Embeddings for Multi-Agent coordination (GEMA), which couples any cooperative learner with a State–Goal Graph Encoder (SGE). The SGE is contrastively pre-trained to embed state graphs in a common metric space. At run time each agent builds the state graph, queries the SGE, and computes a scalar distance to the broadcast goal embedding. This distance is appended to the agent's observation and converted into an intrinsic reward signal, providing the agent with progress information. Experiments on two benchmarks show that GEMA accelerates convergence and boosts team returns, outperforming strong MARL baselines across all scenarios.

1 Introduction

Numerous real-world tasks—ranging from robot formation control to load balancing in data centers—require *teams* of learning agents to steer a system towards a **graph-structured goal**: a desired configuration of entities and their relations. Traditional cooperative multi-agent reinforcement learning (MARL) algorithms face significant challenges in these settings. First, progress toward a goal is often only sparsely rewarded by the environment, which can impede learning. Second, traditional algorithms do not fully exploit the relational structure present in many tasks, missing the permutation-invariant inductive biases that graph-based representations naturally provide. We address these challenges with **Graph Embeddings for Multi-Agent Coordination (GEMA)**. GEMA is a plug-in module that can be paired with any cooperative MARL algorithm. Before policy learning, we contrastively pre-train a **State-Graph Encoder (SGE)** that maps both the current system graph and the objective graph into a metric space calibrated to task progress. At run time, each agent reconstructs the current state graph, feeds both graphs through the frozen SGE, and obtains (i) a similarity feature appended to its private observation, and (ii) an intrinsic reward equal to that similarity.

Our study makes three contributions. First, it formalises graph objectives in cooperative MARL and designs SGE, a contrastively pre-trained module that aligns state and goal graphs in a metric space. Second, it introduces GEMA, which equips agents with progress-aware features and potential-based shaping using the pre-trained SGE without modifying the underlying learner. Third, extensive experiments on *cooperative navigation* and a *load balancing* benchmark environment show that GEMA accelerates convergence, improves asymptotic returns, and scales efficiently from three to ten agents, outperforming baselines such as MAPPO, MASAC, VDN, and MADDPG.

2 Related Work

Early Feudal reinforcement learning (RL) designed low-level controllers rewarded when achieving sub-goal states selected by higher-level managers, establishing the hierarchical perspective on goal conditioning (Dayan and Hinton, 1992). Similarly and independently, although not strictly related to the concept of "goal", potential-based reward shaping demonstrated that adding intrinsic rewards of the form $\Phi(s') - \Phi(s)$ preserves optimality, provided Φ is a well-defined potential over states (NG, 1999). These foundational perspectives laid the groundwork for later methods that treat goals as structured inputs to be encoded and injected directly into the learning process.

Building on this, recent work in goal-conditioned RL has explored how to embed goals into policy and value networks, and how to augment sparse-reward environments with goal-aware replay and training strategies. Universal Value Function Approximators (UVFAs) embed both state and goal and exploit their common structure to calculate a single scalar regression output as value estimate (Schaul et al., 2015). Hindsight Experience Replay exploits goal relabeling to overcome sparse rewards, retrospectively treating each reached state as if it had been the intended goal (Andrychowicz et al., 2017). Feudal Network (FuN) refine the initial intuition of Feudal RL by having higher-level Managers output a latent goal vector g_t every c steps and training it via a cosine-similarity policy-gradient update. At the same time, a lower-level policy Worker receives an intrinsic reward proportional to how closely its trajectory follows those directions (Vezhnevets et al., 2017). While our approach takes inspiration from such foundational methods and how they exploit state-goal structural relations, we pre-train a contrastive encoder that embeds graph representations of state and goal into a metric space calibrated to task progress, and then leverages this distance both as an observation feature and as an intrinsic reward signal for multi-agent domains.

Goal conditioning in cooperative MARL remains comparatively under-explored. Recent work begins to close this gap: Latent Goal Allocation infers per-agent subgoals as latent variables from demonstrations (Chen et al., 2021), MASER generates per-agent subgoals from replay and rewards agents for achieving them while preserving team value consistency (Jeon et al., 2022). LAGMA constructs a latent space via a VQ-VAE and samples goal-reaching trajectories, using distance in that latent space as an intrinsic bonus (Na and Moon, 2024). These methods either infer subgoals, learn individual intrinsic functions, or operate in entire goal-reaching trajectories. Our method assumes that both the desired configuration and the current configuration can be reconstructed as graphs by every agent. A contrastively trained state-goal graph encoder then supplies a global progress-aware similarity that is shared as an additional observation and as an intrinsic reward shaping signal.

3 Preliminaries

Decentralized Markov decision processes. A decentralized Markov decision process (Dec-MDP) extends the standard Markov decision process (MDP) to cooperative multi-agent settings with decentralized information. Formally, a Dec-MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}_{i=1}^{N}, \mathcal{O}_{i=1}^{N}, P, R, \gamma)$, where $\mathcal{S}$ is the global state space, $\mathcal{A}_i$ denotes the action space available to the ith agent, and $\mathcal{O}i$ represents the observation space for agent i. In a Dec-MDP, the global state is determined by the combination of all agents' observations (Bernstein et al., 2002). The transition function $P : \mathcal{S} \times (\mathcal{A}_1 \times \cdots \times \mathcal{A}_N)$ describes state transitions, while $R : \mathcal{S} \times (\mathcal{A}_1 \times \cdots \times \mathcal{A}_N) \to \mathbb{R}$ is a common reward function shared by all agents. The discount factor $\gamma \in [0, 1]$ determines the importance of future rewards. MARL has been employed to tackle Dec-MDP, often through the centralized training with decentralized execution (CTDE) paradigm. In this framework, agents leverage centralized information during training to learn coordinated behaviors.

Graph Neural Networks. Graph neural networks (GNNs) perform differentiable message passing over graph structures, offering permutation-invariant inductive biases beneficial for multi-agent scenarios (Battaglia et al., 2018). In MARL, agents are represented as nodes, while edges capture relationships such as physical proximity or communication links (Jiang et al., 2020). At each layer, nodes iteratively aggregate and update embeddings based on messages from neighbors, effectively

capturing local interactions and higher-order dependencies. Additionally, graph-level embeddings, which summarize the entire graph structure, can be obtained through permutation-invariant pooling functions (e.g., sum, mean, max) applied to node embeddings (Xu et al., 2019).

Contrastive representation learning. Contrastive representation learning is a paradigm for learning embeddings that capture meaningful (dis)similarities between data samples. The core objective is to learn how to map given inputs into a latent space where similar inputs are positioned closer together, while different ones are placed farther apart. This approach often involves comparing pairs or groups of samples to learn discriminative features that reflect the underlying structure of the data. Within this framework falls the triplet loss function, proposed in the context of face recognition (Schroff et al., 2015). It operates on triplets of samples: an anchor x^a, a positive sample x^p (similar to the anchor), and a negative sample x^n (dissimilar to the anchor). The goal is to ensure that the distance between the anchor and the positive is less than the distance between the anchor and the negative by at least a margin α. This is formalized as:

$$\mathcal{L}_{\text{triplet}} = \sum_{i=1}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \tag{1}$$

Here, $f(\cdot)$ denotes the embedding function (often a neural network), and $[\cdot]_+$ represents the hinge function, which outputs the value inside if it's positive and zero otherwise. The margin α enforces a minimum separation between positive and negative pairs in the embedding space.

Multi-Agent Proximal Policy Optimization. Multi-Agent Proximal Policy Optimization (MAPPO) is an extension of the Proximal Policy Optimization (PPO) algorithm tailored for MARL (Yu et al., 2021). It employs a CTDE framework, where each agent acts using its policy network based on its local observations, i.e. the actor network, while a centralized value function, the critic, is trained using a broader observation often including the entire global state of the environment.

4 Method

Our proposed method introduces a novel and scalable approach to cooperative decentralized multi-agent tasks where a desired system state (i.e., the goal) and the current state can be represented as graphs. The intuition is to learn a latent representation space where the relational similarity of the current configuration and the desired goal can be meaningfully compared, and inform the agents of such (dis)similarities as they train their distributed policy. To this end, prior to policy training, we introduce a contrastive representation learning phase to train a SGE that captures relevant relational properties of the task. Once learned, we employ the SGE to allow each agent to compute an embedding of the current state and measure its similarity to the desired goal. This similarity serves two purposes: (i) it is appended as an additional observation feature available to the agents' policy networks, and (ii) it defines an intrinsic reward signal that complements the environment reward, thereby shaping the learning process toward satisfying the global objective.

In this section, we will first discuss how graphs are constructed from an agent perspective, including the necessary notation. Then we will discuss the generation of state embeddings and how to train the SGE. Finally, we will introduce the complete GEMA architecture employed by each agent.

4.1 State and Goal Graphs

A *state graph* $G = (V, E)$ is an undirected, attributed graph that encodes one joint configuration of the multi-agent system.

- **Nodes.** Each node $v \in V$ corresponds to an entity (e.g. an agent or any other relevant entity to the task) and carries a feature vector $x_v \in \mathbb{R}^{d_x}$ describing its local properties, e.g. position, load, battery level, etc.

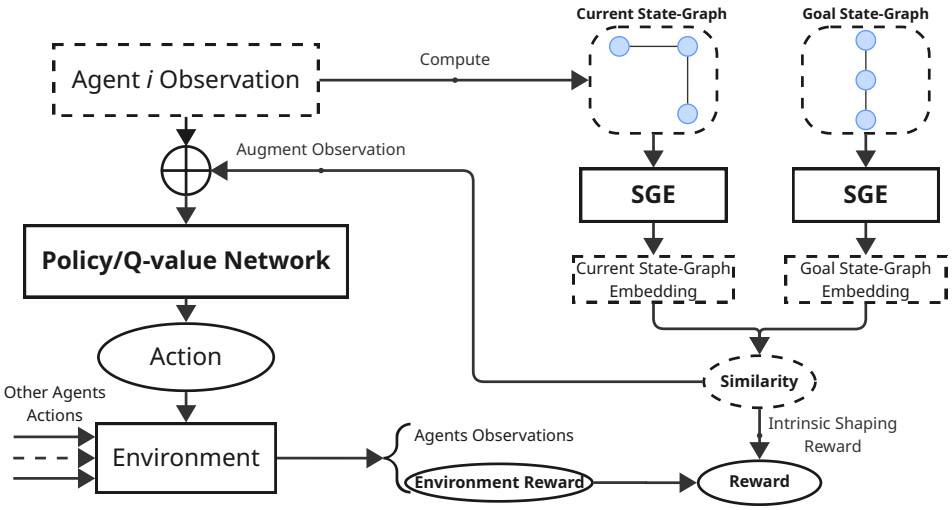

Figure 1: Overview of GEMA. Agents encode local state and shared goal graphs via a pre-trained SGE, their embedding similarity provides both an observation feature and an intrinsic reward.

- **Edges.** Optional edge attributes $e_{uv} \in \mathbb{R}^{d_e}$ capture pairwise relations such as Euclidean distance, line-of-sight, or communication cost. If no explicit metric is provided, we set $e_{uv} = \varnothing$ and learn purely from node attributes.
- **Goal Graph.** A *goal state* is represented by a graph $G^\star$ encoding the desired configuration that the agents must cooperate to achieve. For tasks that admit many interchangeable goals (e.g. any permutation of agents on landmarks), we uniformly sample a single valid instance.

We assume that the environment exposes enough information in each agent's observation o_i to construct the current state graph G. This assumption holds for all benchmarks in Sec. 5.1 and is critical for our proposed method, as every agent must be able to reconstruct the same graphs without additional communication.

4.2 GEMA

4.2.1 Learning the State-Graph Encoder (SGE)

Training the SGE is conducted purely in an offline process where no policies are learned at this stage, and the environment is queried only to record diverse states. We first generate a dataset of state–goal graph pairs annotated with coarse progress labels. Next, an encoder architecture based on GNNs embeds each graph into a permutation-invariant representation. Finally, a contrastive objective with adaptive margins learn the latent space so that the cosine distance faithfully mirrors task progress. The encoder parameters obtained at the end of this phase are frozen and reused during policy optimisation.

Dataset. We collect a replay buffer $\mathcal{D} = \{(G_k, G_k^\star)\}_{k=1}^{|\mathcal{D}|}$ by rolling out random interactions, simple heuristics, or expert controllers chosen only for their state-space coverage. For each entry we also instantiate a goal graph $G_k^\star$ that captures the configuration the team should achieve, without encoding any agent–to–role assignment. In a landmark-covering task, for instance, $G_k^\star$ contains one agent node per landmark but does not specify which particular agent must occupy which landmark. Each pair $(G_k, G_k^\star)$ is then assigned a *progress label* $c_k \in \{1, \dots, C\}$ that quantifies closeness to the goal via task-specific heuristics, such as geometric error thresholds or the number of satisfied constraints.

Graph Neural Network. State and goal graphs are embedded with an L-layer GNN followed by a pooling layer:

$$h_v^{(\ell)} = \phi_{\text{upd}}\Big(h_v^{(\ell-1)}, \bigoplus_{u\in\mathcal{N}(v)} \psi\big(h_v^{(\ell-1)}, h_u^{(\ell-1)}, e_{uv}\big)\Big), \quad f_\theta(G) = \mathcal{R}(h_i^{(\ell)} | v_i \in V),$$

where $\mathcal{N}(v)$ denotes the immediate neighbors of v. ϕ_{upd}, ψ are multilayer perceptrons (MLPs) that share parameters across all layers and nodes, and we denote their union by θ. Message passing and the multiple layers work as relational kernels to provide multi-hop relational context to each node (Jiang et al., 2020). The final pooling layer function $\mathcal{R}$ and the aggregation operation $\bigoplus$ must be permutation-invariant, such that any reordering of node indices leaves $f_\theta(G)$ unchanged. Optionally, raw node features and the final $f_\theta(G)$ representation can be further transformed by additional MLPs.

Contrastive representation learning with adaptive margins. We sample graph triplets (G^a, G^p, G^n) from the replay buffer such that the anchor and positive share the same progress label $c(G^a) = c(G^p)$, while the negative has $c(G^n) \neq c(G^a)$. Each graph is processed by the SGE and a comparison MLP, producing embeddings $z^a = f_\theta(G^a)$, $z^p = f_\theta(G^p)$, and $z^n = f_\theta(G^n)$. We minimise the cosine-based triplet loss

$$d_{\cos}(u, v) = \frac{1}{2}(1 - \frac{u^\top v}{\|u\|_2 \, \|v\|_2}), \mathcal{L}_{\text{triplet}} = \Big[d_{\cos}\big(z^a, z^p\big) - d_{\cos}\big(z^a, z^n\big) + \alpha_{pn}\Big]_+,$$

where adaptive margin α_{pn} is defined as $\alpha_{pn} = m\,|c(G^a) - c(G^n)|$, with $m > 0$ as a global scale factor, and grows linearly with the class gap, driving embeddings of states that differ more in task progress to be proportionally farther apart under the cosine metric. The cosine distance d_{cos} is multiplied by $\frac{1}{2}$ limiting its range to $[0, 1]$. Once the loss converges, the encoder parameters θ_{sge} are frozen, keeping the learned representation stable during policy updates.

4.2.2 Training the Agents

We integrate the SGE into the online learning loop by augmenting each agent's perception with (i) a shared, task-level context derived from the current state–goal similarity, and (ii) an intrinsic reward proportional to that similarity. Figure 1 illustrates the overall flow of our proposed method from the point of view of a single agent.

At every time step, each agent constructs the current state graph G_t and goal graph $G^\star$. The SGE generates embeddings $z_t = f_{\theta_{\text{sge}}}(G_t)$ and $z^\star = f_{\theta_{\text{sge}}}(G^\star)$. We then compute their cosine similarity $c_t = \cos(z_t, z^\star)$, serving as a compact, global context shared among all agents. Each agent i then concatenates c_t with its observation $o_{t,i}$ and feds it into a policy/ or -function head. The scalar similarity c_t provides an intrinsic shaping reward $r_t^{\text{int}} = c_t$, combined with the environment reward as $\tilde{r}_t = r_t^{\text{env}} + r_t^{\text{int}}$. Our method is algorithm-agnostic and can be adopted by policy-based and value-based learning approaches, as it simply involves adding the cosine similarity between embeddings from the SGE into the agent's observation and guiding the agents through intrinsic reward signals. Additional experimental implementation details are provided in Section 5.2.2.

5 Experiments

In this section, we evaluate the performance of GEMA across two cooperative multi-agent tasks: *cooperative navigation* and *load balancing*. We compare GEMA against four widely adopted MARL baselines: VDN(Sunehag et al., 2018), MAPPO(Yu et al., 2021), MADDPG(Lowe et al., 2017a), and a similar extension for multi-agent environments for SAC (Haarnoja et al., 2018), covering both value-based and policy-based paradigms. We report performance in terms of episodic returns and task-specific success metrics, and analyze scalability as the number of agents increases.

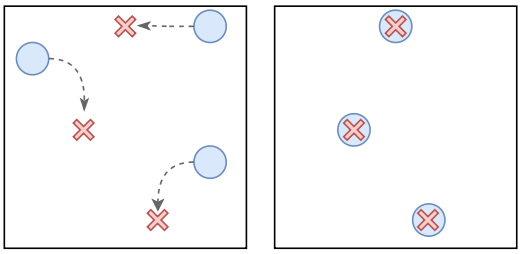

Figure 2: Cooperative navigation environment.

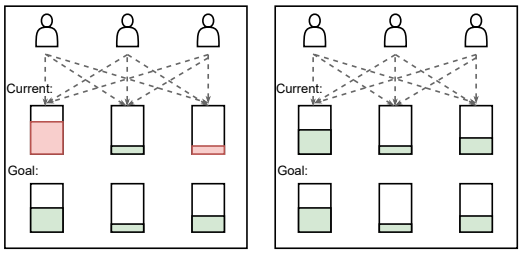

Figure 3: Load balancing environment.

5.1 Benchmark Environments

We evaluate GEMA's performance in two benchmark environments with different goal structures and reward density, these being *cooperative navigation* and *load balancing*. Both tasks are fully observable, allowing each agent to encode the complete state, and the goal configuration is known as part of the agents' observation.

Cooperative navigation. In the cooperative navigation task, N agents move within a 4×4 two-dimensional map to cover N landmarks (Lowe et al., 2017b). We model the environment as a fully connected graph with two node types: agent nodes and landmark nodes. Each node is represented by its 2-D position and a binary type indicator (0 for agents, 1 for landmarks). Every edge carries two features: the Euclidean distance and the unit direction vector between the connected nodes. Agents receive a shared reward that increases as the summed distances between agents and landmarks decrease; the closer each agent is to a distinct landmark, the higher the collective return. The reward is linearly scaled to lie within $[-1, 1]$.

Load balancing. In the load balancing task, three cooperative agents each control a dedicated computing resource. A resource's load is the number of jobs it is currently processing; accepting or forwarding a job increments that load. The agents aim to redistribute jobs so that the loads match a desired target-load vector. The environment is represented as a fully connected graph with two node types: agent nodes and target nodes. Each node is characterised by its load value and a binary type indicator (0 for agents, 1 for targets). Every edge carries the load difference between connected nodes. At the beginning of every episode the environment samples the current-load vector $L \in \mathbb{R}^3$ and the target-load vector $T \in \mathbb{R}^3$; both are normalised so that $\sum_i L_i = 1$ and $\sum_i T_i = 1$. At each timestep, an agent receives a new job and chooses an action $a_t \in \{0, 1, 2\}$ that specifies the resource to which the job is routed. After every action, the environment produces a reward common to all agents, proportional to the number of targets satisfied. Because the targets are unordered, any agent may fulfil any target, but no target may be satisfied by more than one agent. The reward is linearly scaled to lie within $[-1, 1]$.

5.2 Experimental settings

5.2.1 SGE Dataset Generation

Cooperative navigation. We build the dataset with a heuristic that first computes the pairwise distances between agents and landmarks, then applies the Hungarian algorithm (Kuhn, 1955) to assign each agent to a landmark. To diversify the samples, we introduce a 20% probability that an agent selects a random action. Each sample is assigned to one of ten distance classes defined by the upper-bound thresholds $\{0.1, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, \infty\}$. Let d be the sum of the agents' minimum Euclidean distances to the landmarks. A state belongs to class k if d is smaller than the k-th threshold. The final dataset contains $\approx 9 \times 10^5$ samples, allocated uniformly at $\approx 1 \times 10^5$ instances per class. The margin α_{pn} grows linearly at a rate of 0.1 per class.

Load balancing. We generate the dataset with a deficit-oriented heuristic, Algorithm 1. Each resource is assigned a target load; at every timestep, each agent forwards its job to the node with the greatest deficit—that is, the resource most under-loaded relative to its target. Every environment state is labelled with one of three classes: (0) no targets satisfied, (1) exactly one target satisfied, and (2) at least two targets satisfied. Because $\sum_i T_i = 1$, satisfying two targets automatically satisfies the third. The final dataset contains $\approx 4.5 \times 10^5$ samples, allocated uniformly at $\approx 1.5 \times 10^5$ instances per class. The margin α_{pn} grows linearly at a rate of 0.3 per class.

Algorithm 1: PICKJOBDESTINATION($\mathbf{L}, \mathbf{T}$)

Input: Current loads $\mathbf{L} \in \mathbb{R}^3$, target loads $\mathbf{T} \in \mathbb{R}^3$
Output: Destination index $k \in \{0, 1, 2\}$
$\mathit{total} \leftarrow \sum_{i=0}^{2} L_i$;
for $i \leftarrow 0$ **to** 2 **do**
  $\mathit{ratio}_i \leftarrow \dfrac{L_i}{\mathit{total}}$;
  $\mathit{deficit}_i \leftarrow T_i - \mathit{ratio}_i$;
$k \leftarrow \arg\max_{i \in \{0,1,2\}} \mathit{deficit}_i$;
return k;

5.2.2 GEMA, Policy, and Value Networks

Our SGE architecture consists of three stages: (i) a two-layer input MLP that embeds raw node features, (ii) a GATv2 (Brody et al., 2022) layer applied over the entire graph to produce contextualized node embeddings, and (iii) a projection MLP used exclusively during contrastive pretraining and discarded afterward. GEMA is instantiated using MAPPO as the underlying algorithm. Its actor network employs a single GATv2 convolution, which processes the agent graph. The resulting embedding of the agent node is appended with the cosine similarity between current and goal state embeddings as computed by the SGE. Finally, this representation is fed into a dedicated MLP to produce the action distribution. The critic concatenates all the node-level features of the raw state graph and appends the current state and goal embeddings, along with their cosine similarity.

To ensure a fair comparison, all baselines are evaluated under both GNN-based and MLP-based as policy/value networks. In the MLP version, we use a flattened private observation with full state information relative to each observing agent as implemented in Lowe et al. (2017a) for the cooperative navigation task. Additional information on the implementation of the architectures involved, as well as the hyperparameters adopted, can be found in Appendix A.

5.3 Results

Cooperative Navigation. Figure 4 shows the mean episodic reward over training when every method employs a GNN actor. GEMA achieves a return of 0.936 and converges with fewer samples than any baseline. MAPPO follows with 0.859, then VDN with 0.795, MASAC with 0.770, and MADDPG with 0.320. Figure 5 presents the same task when the baselines use an MLP actor, whereas GEMA retains its GNN actor. In this setting, MAPPO attains a mean return of 0.872, MASAC 0.865, VDN 0.790, and MADDPG 0.550.

Table 1 reports the mean episodic return ($\pm$ standard deviation) in cooperative navigation as the team size increases from three to ten agents. For each setting, we ran 200 evaluation episodes under seven random seeds, comparing the best-performing GEMA model with the best-performing GNN-based

Table 1: Average return (mean $\pm$ std) across team sizes in cooperative navigation.

Algorithm	3 Agents	6 Agents	10 Agents
GEMA	**93.79 ± 2.58**	**93.48 ± 1.47**	**93.08 ± 1.04**
MAPPO	86.59 ± 4.00	88.51 ± 2.20	89.69 ± 1.49
MASAC	77.05 ± 7.92	84.07 ± 4.18	87.77 ± 2.62
VDN	81.18 ± 5.98	84.81 ± 3.39	86.65 ± 2.51
MADDPG	30.62 ± 11.27	36.70 ± 7.87	40.78 ± 5.99

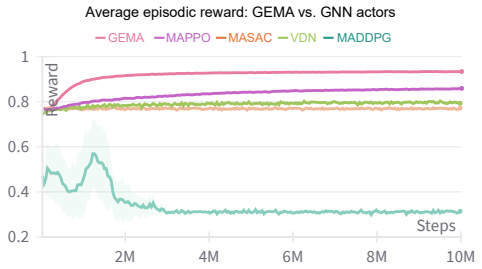

Figure 4: Average episodic reward on cooperative navigation using GNN actors.

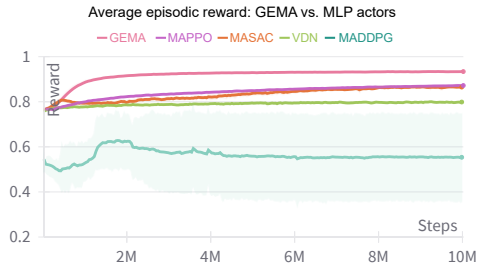

Figure 5: Average episodic reward on cooperative navigation: GEMA (GNN actor) vs. MLP-based.

MAPPO, MASAC, VDN, and MADDPG. GEMA consistently achieves the highest performance, maintaining scores above 93 with minimal variance. MAPPO ranks second but trails GEMA by 4–7 points across all team sizes. MASAC and VDN returns remain 5–16 points lower than GEMA's at every scale. MADDPG performs worst, remaining well below the other methods.

Load Balancing. Figure 6 plots the mean episodic reward during training, averaged over five random seeds. As in cooperative navigation, GEMA converges more rapidly than the baselines. By the end of training, it reaches an average return of -0.008, whereas MAPPO and VDN stabilise at -0.158 and -0.396, respectively. Table 2 summarises the evaluation results. We report the mean episodic return ($\pm$ standard deviation) and the average proportion of timesteps in which the load-balancing constraints are satisfied. Each metric is computed over 100 evaluation episodes and seven random seeds. GEMA secures the only positive mean reward, 0.10 ± 0.59, outperforming MAPPO (-0.04 ± 0.62) and VDN (-0.23 ± 0.51). It also maintains a feasible load distribution for 29.5% of the timesteps, compared with 24.0% for MAPPO and 13.2% for VDN.

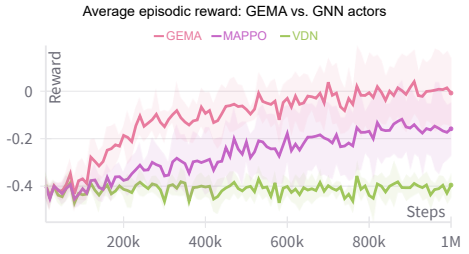

Figure 6: Average episodic reward on load balancing using GNN actors.

6 Conclusion and Future Work

We introduced GEMA, a plug-in that lets cooperative MARL algorithms exploit the symbolic and structured information carried by graph objectives via a pre-trained SGE. Its similarity score is included in the agent observations and incorporated as an intrinsic shaping reward, accelerating convergence and improving final performance on both dense and sparse reward benchmarks while scaling from three to ten agents and outperforming state-of-the-art MARL algorithms. Future research will explore end-to-end training of the encoder, extensions to partial observability, hierarchical subgoal discovery, and deployment on environments where state reconstruction is imperfect.

Table 2: Average return and no. of steps where load balancing constraints are satisfied.

Algorithm	Avg. Reward (mean $\pm$ std)	Steps (mean $\pm$ std)
GEMA	**0.10 $\pm$ 0.59**	**29.5 $\pm$ 38.9**
MAPPO	-0.04 $\pm$ 0.62	24.0 $\pm$ 36.5
VDN	-0.23 $\pm$ 0.51	13.2 $\pm$ 25.8

References

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5055–5065, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. URL http://arxiv.org/abs/1806.01261.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3690469.

S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks?, 2022. URL https://arxiv.org/abs/2105.14491.

R. Chen, P. Huang, and L. Shi. Latent goal allocation for multi-agent goal-conditioned self-supervised imitation learning. *Bayesian Deep Learning - NeurIPS Workshop*, 2021.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

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

J. Jeon, W. Kim, W. Jung, and Y. Sung. Maser: Multi-agent reinforcement learning with subgoals generated from experience replay buffer. In *International conference on machine learning*, pages 10041–10052. PMLR, 2022.

J. Jiang, C. Dun, T. Huang, and Z. Lu. Graph convolutional reinforcement learning, 2020. URL https://arxiv.org/abs/1810.09202.

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi: https://doi.org/10.1002/nav.3800020109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6382–6393, Red Hook, NY, USA, 2017a. Curran Associates Inc. ISBN 9781510860964.

R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017b. URL http://arxiv.org/abs/1706.02275.

H. Na and I.-C. Moon. Lagma: latent goal-guided multi-agent reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

A. NG. Policy invariance under reward transformation: Theory and application to reward shaping. In *Proc. 16th Int. Conf. on Machine Learning, 1999*, 1999.

T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 815–823. IEEE, June 2015. doi: 10.1109/cvpr.2015.7298682. URL http://dx.doi.org/10.1109/CVPR.2015.7298682.

P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3540–3549. JMLR.org, 2017.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

A Extended Experimental Setup

A.1 GEMA Implementation

A.1.1 SGE

The SGE comprises three distinct stages:

1. **Input Embedding:** Each node's raw features are first embedded using a two-layer MLP. Specifically, the input encoder consists of 128 hidden units and produces a 16-dimensional output vector, forming the initial representation of nodes for the subsequent graph encoder.
2. **Graph Encoder:** We employ a GATv2 convolutional layer (Brody et al., 2022) with multi-head attention. The number of attention heads and dimensionality of node embeddings depend on the task:
   - *Cooperative navigation:* one attention head yielding 32-dimensional node embeddings.
   - *Load balancing:* three attention heads producing 48-dimensional node embeddings.

   A sum-pooling operation aggregates these embeddings across all nodes to form a graph-level representation.
3. **Contrastive Head:** The pooled graph embedding is further transformed by a two-layer MLP, consisting of 64 hidden units followed by a 16-dimensional output layer. This module is only present during the SGE training phase and dropped thereafter.

A.1.2 Actor

Let's assume agent i corresponds to node i in the graph:

1. Raw node features are embedded using a two-layer MLP with a 128-unit hidden layer, outputting 16-dimensional node embeddings.
2. These embeddings feed into a three-head GATv2 convolutional layer, resulting in 32-dimensional node-level representations, resulting in a single message passing round.
3. We append the similarity between state and goal graph embeddings as produced by the SGE to the new representation of agent i features.
4. Agent i processes its node representation (33-dim) using a three-layer MLP with two hidden layers of 256 units each, resulting in the action logits for policy-based methods or Q-values for value-based methods.

A.1.3 Critic

1. We concatenate all the node-level features of the raw state graph.
2. We compute embeddings z_t and z^* for the current state and goal using the frozen SGE.
3. We compute the cosine similarity $c_t = \cos(z_t, z^*)$.
4. The critic receives as input the concatenation of the node-level features, the scalar c_t alongside the two embeddings.
5. A two-layer MLP with a single hidden layer of 256 units processes this compact representation to produce the scalar state-value estimate.

A.2 Baseline Architectures

For a fair evaluation, we tested each baseline algorithm using two distinct parameterizations, adopting a GNN and an MLP for policy actors/Q-functions.

A.2.1 Actors

GNN Actor Architecture. Let's assume agent i corresponds to node i in the graph:

1. Raw node features are embedded using a two-layer MLP with a 128-unit hidden layer, outputting 16-dimensional node embeddings.
2. These embeddings feed into a three-head GATv2 convolutional layer, resulting in 32-dimensional node-level representations, resulting in a single message passing round.
3. Agent i processes its node representation using a three-layer MLP with two hidden layers of 256 units.

MLP Actor Architecture. The MLP-based actor processes a flattened private observation with full state information relative to each observing agent via a simpler architecture:

1. This observation is processed by a two-layer MLP featuring a single hidden layer of 256 units.
2. The MLP directly outputs action logits or Q-values.

A.2.2 Critics (for Policy-based methods)

All policy-based methods employ a shared critic with parameters common to all agents.

1. We concatenate all the agents' node-level features in case the observation has a graph structure (algorithm adopting GNN actors) or the agents' private observations otherwise.
2. A two-layer MLP with a single hidden layer of 256 units processes this compact representation to produce the scalar state-value estimate.

A.3 Training hyperparameters

Training seeds: 3412, 0, 45455, 566778, 445
Evaluation seeds: 225131, 11225, 22355, 99985, 113374, 998653, 11253

Table 3: Key hyper-parameters for COOPERATIVE NAVIGATION.

Hyper-parameter	**Value**
γ	0.9
Learning rate	5×10^{-5}
Adam ϵ	1×10^{-6}
Grad. clip (norm)	Yes, 5
Target update	Soft ($\tau = 0.005$)
ε start$\rightarrow$end	0.8$\rightarrow$0.01
Anneal frames	1 000 000
Total frames	10 000 000
On-policy	
Frames / batch	60 000
Minibatch iters	45
Minibatch size	4 096
Off-policy	
Frames / batch	6 000
Opt. steps / batch	1 000
Train batch size	128
Buffer size	1 000 000

Table 4: Key hyper-parameters for LOAD BALANCING.

Hyper-parameter	**Value**
γ	0.9
Learning rate	5×10^{-5}
Adam ϵ	1×10^{-6}
Grad. clip (norm)	Yes, 5
Target update	Soft ($\tau = 0.005$)
ε start$\rightarrow$end	0.8$\rightarrow$0.01
Anneal frames	300 000
Total frames	1 000 000
On-policy	
Frames / batch	10 000
Minibatch iters	45
Minibatch size	4 096
Off-policy	
Frames / batch	10 000
Opt. steps / batch	100
Train batch size	100
Buffer size	100 000