Value-Based Deep Multi-Agent Reinforcement Learning with Dynamic Sparse Training

Pihe Hu[*]
Tsinghua University
Beijing, China
hupihe@gmail.com

Shaolong Li[*]
Central South University
Changsha, China
shaolongli16@gmail.com

Zhuoran Li
Tsinghua University
Beijing, China
lizr20@mails.tsinghua.edu.cn

Ling Pan
Hong Kong University of
Science and Technology
Hong Kong, China
lingpan@ust.hk

Longbo Huang[†]
Tsinghua University
Beijing, China
longbohuang@tsinghua.edu.cn

Abstract

Deep Multi-agent Reinforcement Learning (MARL) relies on neural networks with numerous parameters in multi-agent scenarios, often incurring substantial computational overhead. Consequently, there is an urgent need to expedite training and enable model compression in MARL. This paper proposes the utilization of dynamic sparse training (DST), a technique proven effective in deep supervised learning tasks, to alleviate the computational burdens in MARL training. However, a direct adoption of DST fails to yield satisfactory MARL agents, leading to breakdowns in value learning within deep sparse value-based MARL models. Motivated by this challenge, we introduce an innovative Multi-Agent Sparse Training (MAST) framework aimed at simultaneously enhancing the reliability of learning targets and the rationality of sample distribution to improve value learning in sparse models. Specifically, MAST incorporates the Soft Mellowmax Operator with a hybrid TD-(λ) schema to establish dependable learning targets. Additionally, it employs a dual replay buffer mechanism to enhance the distribution of training samples. Building upon these aspects, MAST utilizes gradient-based topology evolution to exclusively train multiple MARL agents using sparse networks. Our comprehensive experimental investigation across various value-based MARL algorithms on multiple benchmarks demonstrates, for the first time, significant reductions in redundancy of up to $20\times$ in Floating Point Operations (FLOPs) for both training and inference, with less than 3% performance degradation.

1 Introduction

Multi-agent reinforcement learning (MARL) [Shoham and Leyton-Brown, 2008], coupled with deep neural networks, has not only revolutionized artificial intelligence but also showcased remarkable success across a wide range of critical applications. From mastering multi-agent video games such as Quake III Arena [Jaderberg et al., 2019], StarCraft II [Mathieu et al., 2021], Dota 2 [Berner et al., 2019], and Hide and Seek [Baker et al., 2020] to guiding autonomous robots through intricate real-world environments [Shalev-Shwartz et al., 2016, Da Silva et al., 2017, Chen et al., 2020b], deep MARL has emerged as an indispensable and versatile tool for addressing complex, multifaceted challenges. Its unique capability to capture intricate interactions and dependencies among multiple

[*]Equal contribution.
[†]Corresponding author.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

agents has spurred novel solutions, solidifying its position as a transformative paradigm across various domains [Zhang et al., 2021, Albrecht et al., 2023].

However, the exceptional success of deep MARL comes at a considerable computational cost. Training these agents involves the intricate task of adapting neural networks to accommodate an expanded parameter space, especially in scenarios with a substantial number of agents. For instance, the training regimen for AlphaStar [Mathieu et al., 2021], tailored for StarCraft II, extended over a grueling 14-day period, employing 16 TPUs per agent. Similarly, the OpenAI Five [Berner et al., 2019] model for Dota 2 underwent an extensive training cycle spanning 180 days and harnessing thousands of GPUs. This exponential increase in computational demands as the number of agents grows (and the corresponding joint action and state spaces) poses a significant challenge during MARL deployment.

To tackle these computational challenges, researchers have delved into dynamic sparse training (DST), a method that trains neural network models with dynamically sparse topology. For example, RigL [Evci et al., 2020] can train a 90%-sparse network from scratch in deep supervised learning without performance degradation. However, in deep reinforcement learning (DRL), the learning target evolves in a bootstrapping manner [Tesauro et al., 1995], and the distribution of training data is path-dependent [Desai et al., 2019], posing additional challenges to sparse training. Improper sparsification can result in irreversible damage to the learning path [Igl et al., 2020]. Initial attempts at DST in sparse single-agent DRL training have faced difficulties in achieving consistent model compression across diverse environments, as documented in [Sokar et al., 2022, Graesser et al., 2022]. This is mainly because sparse models may introduce significant bias, leading to unreliable learning targets and exacerbating training instability as agents learn through bootstrapping. Moreover, the partially observable nature of each agent makes training non-stationarity inherently more severe in multi-agent settings. Collectively, these factors pose significant challenges for value learning in each agent under sparse models.

We present a motivating experiment in Figure 1, where we evaluated various sparse training methods on the `3s5z` task from SMAC [Samvelyan et al., 2019] using a neural network with only 10% of its original parameters. Classical DST methods, including SET [Mocanu et al., 2018] and RigL [Evci et al., 2020], demonstrate poor performance in MARL scenarios, along with using static sparse networks (SS). Additionally, RLx2 as proposed in [Tan et al., 2022] proves ineffective for multi-agent settings, despite enabling DST for single-agent settings with 90% sparsity. In contrast, our MAST framework in this work achieves a win rate of over 90%. In addition, the only prior attempt to train sparse MARL agents, as described in [Yang et al., 2022], prunes agent networks during training with weight grouping [Wang et al., 2019]. However, this approach fails to maintain sparsity throughout training, and only achieves a final model sparsity of only 80%. Moreover, their experiment is confined to a two-user environment, PredatorPrey-v2, in MuJoCo [Todorov et al., 2012]. These observations highlight the fact that, despite its potential, the application of sparse networks in the context of MARL remains largely unexplored (A comprehensive literature review is deferred in Appendix A.1). Consequently, a critical and intriguing question arises:

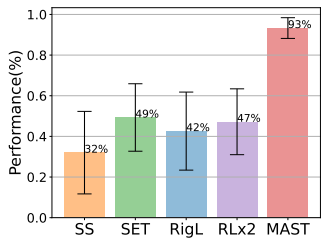


Figure 1: Comparison of different sparse training methods.

Can we train MARL agents effectively using ultra-sparse networks throughout?

We affirmatively address the question by introducing a novel sparse training framework, Multi-Agent Sparse Training (MAST). Since improper sparsification results in network fitting errors in the learning targets and incurs large policy inconsistency errors in the training samples, MAST ingeniously integrates the Soft Mellowmax Operator with a hybrid TD-(λ) schema to establish reliable learning targets. Additionally, it incorporates a novel dual replay buffer mechanism to enhance the distribution of training samples. Leveraging these components, MAST employs gradient-based topology evolution to exclusively train multiple MARL agents using sparse networks. Consequently, MAST facilitates the training of highly efficient MARL agents with minimal performance compromise, employing ultra-sparse networks throughout the training process.

Our extensive experimental investigation across various value-based MARL algorithms on multiple SMAC benchmarks reveals MAST's ability to achieve model compression ranging from $5\times$ to $20\times$, while incurring minimal performance trade-offs (under 3%). Moreover, MAST demonstrates an

impressive capability to reduce the Floating Point Operations (FLOPs) required for both training and inference by up to 20×, showcasing a significant margin over other baselines (detailed in Section 4).

2 Preliminaries

Deep MARL We model the MARL problem as a decentralized partially observable Markov decision process (Dec-POMDP) [Oliehoek et al., 2016], represented by a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, \gamma \rangle$. Deep Multi-Agent Q-learning extends the deep Q learning method [Mnih et al., 2013] to multi-agent scenarios [Sunehag et al., 2018, Rashid et al., 2020b, Son et al., 2019]. The agent-wise Q function is defined over its history τ_i as Q_i for agent i. Subsequently, the joint action-value function $Q_{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})$ operates over the joint action-observation history $\boldsymbol{\tau}$ and joint action $\boldsymbol{u}$. The objective, given transitions $(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}')$ sampled from the experience replay buffer $\mathcal{B}$, is to minimize the mean squared error loss $\mathcal{L}(\theta)$ on the temporal-difference (TD) error $\delta = y - Q_{\text{tot}}(\boldsymbol{\tau}, \boldsymbol{u})$. Here, the TD target $y = r + \gamma \max_{\boldsymbol{u}'} \bar{Q}_{\text{tot}}(\boldsymbol{\tau}', \boldsymbol{u}')$, where $\bar{Q}_{\text{tot}}$ is the target network for the joint action Q-function, periodically copied from Q_{tot}. Parameters of Q_{tot} are updated using $\theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$, with α representing the learning rate.

We focus on algorithms that adhere to the Centralized Training with Decentralized Execution (CTDE) paradigm [Kraemer and Banerjee, 2016], within which, agents undergo centralized training, where the complete action-observation history and global state are available. However, during execution, they are constrained to individual local action-observation histories. To efficiently implement CTDE, the Individual-Global-Maximum (IGM) property [Son et al., 2019] in Eq. (1), serves as a key mechanism:

$$\arg\max_{\boldsymbol{u}} Q_{\text{tot}}(s, \boldsymbol{u}) = \left(\arg\max_{u_1} Q_1(s, u_1), \cdots, \arg\max_{u_N} Q_N(s, u_N) \right). \tag{1}$$

Many deep MARL algorithms [Yu et al., 2022, Pan et al., 2022] adhere to the IGM criterion, such as the QMIX series algorithms [Rashid et al., 2020b,a, Pan et al., 2021]. These algorithms employ a mixing network f_s with non-negative weights, enabling the joint Q-function to be expressed as $Q_{\text{tot}}(s, \boldsymbol{u}) = f_s(Q_1(s, u_1), \cdots, Q_N(s, u_N))$.

Dynamic Sparse Training Dynamic sparse training (DST), initially proposed in deep supervised learning, can train a 90% sparse network without performance degradation from scratch, such as in ResNet-50 [He et al., 2016] and MobileNet [Howard et al., 2017]. In DST, the dense network is randomly sparsified at initialization, as shown in Figure 2, and its topology is dynamically changed during training by link dropping and growing. Specifically, the topology evolution mechanism in MAST follows the RigL method [Evci et al., 2020], which improves the optimization of sparse neural networks by leveraging weight magnitude and gradient information to jointly optimize model parameters and connectivity.

Figure 2: Illustration of dynamic sparse training.

RigL periodically and dynamically drops a subset of existing connections with the smallest absolute weight values and concurrently grows an equivalent number of empty connections with the largest gradients. The pseudo-code of RigL is given in Algorithm 1, where the symbol $\odot$ denotes the element-wise multiplication operator, M_θ symbolizes the binary mask that delineates the sparse topology for the network θ, and ζ_t is the update fraction in training step t. This process maintains the network sparsity throughout the training with a strong evolutionary ability that saves training FLOPs and gives a sparse model after training. However, in DRL, the learning target evolves in a bootstrapping manner [Tesauro et al., 1995], such that the distribution of training data is path-dependent [Desai et al., 2019], posing additional challenges to sparse training. Moreover, the partially observable nature of each agent exacerbates training non-stationarity, particularly in multi-agent settings. These factors collectively present significant hurdles for value learning in each agent under sparse models. As illustrated in Figure 1,

Algorithm 1 Topology Evoltion[Evci et al., 2020]

1: θ_l, N_l, s_l: parameters, number of parameters, sparsity of layer l.
2: **for** each layer l **do**
3: $k = \zeta_t(1 - s_l)N_l$
4: $\mathbb{I}_{\text{drop}} = \text{ArgTopK}(-|\theta_l \odot M_{\theta_l}|, k)$
5: $\mathbb{I}_{\text{grow}} = \text{ArgTopK}_{i \notin \theta_l \odot M_{\theta_l} \setminus \mathbb{I}_{\text{drop}}}(|\nabla_{\theta_l} L|, k)$
6: Update M_{θ_l} according to $\mathbb{I}_{\text{drop}}$ and $\mathbb{I}_{\text{grow}}$
7: $\theta_l \leftarrow \theta_l \odot M_{\theta_l}$
8: **end for**

attempts to train ultra-sparse MARL models using simplistic topology evolution or the sparse training framework for single-agent RL have failed to achieve satisfactory performance. Therefore, MAST introduces innovative solutions to enhance value learning in ultra-sparse models by simultaneously improving the reliability of learning targets and the rationality of sample distribution.

3 Enhancing Value Learning in Sparse Models

This section outlines the pivotal components of the MAST framework for training sparse MARL agents. MAST introduces innovative solutions to enhance the accuracy of value learning in ultra-sparse models by concurrently refining training data targets and distributions. Consequently, the topology evolution in MAST effectively identifies appropriate ultra-sparse network topologies. This approach aligns with single-agent DRL, where sparse training necessitates co-design with the value learning method as described in [Tan et al., 2022]. However, the partially observable nature of each agent exacerbates training non-stationarity in multi-agent settings. As illustrated in Figure 3, MAST implements two key innovations to achieve accurate value learning in ultra-sparse models: i) hybrid TD(λ) targets combined with the Soft Mellowmax operator to mitigate estimation errors arising from network sparsity, and ii) dual replay buffers to reduce policy inconsistency errors due to sparsification.

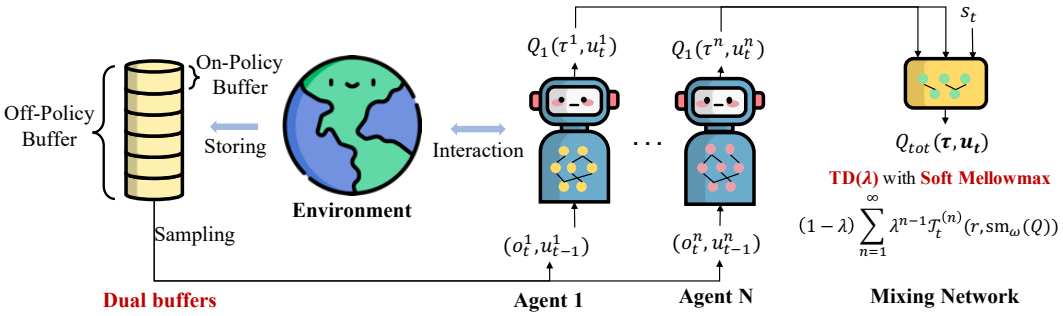


Figure 3: An example of the MAST framework based on QMIX.

3.1 Improving the Reliability of Training Targets

Initially, we observe that the expected error is amplified under sparse models, motivating the introduction of multi-step TD targets. However, different environments may require varying values of step lengths to achieve optimal performance. Consequently, we focus on **hybrid TD(λ) targets**, which can achieve performance comparable to the best step length across different settings. Additionally, we find that the overestimation problem remains significant in sparse models. To address this, we propose the use of the **Soft Mellowmax operator** in constructing learning targets. This operator is effective in reducing overestimation bias without incurring additional computational costs.

Hybrid TD(λ) Targets. In deep multi-agent Q-learning, temporal difference (TD) learning is a fundamental method for finding an optimal policy, where the joint action-value network is iteratively updated by minimizing a squared loss driven by the TD target. Let M_θ be a binary mask representing the network's sparse topology, and denote the sparse network as $\hat{\theta} = \theta \odot M_\theta$, where $\odot$ signifies element-wise multiplication. Since sparse networks operate within a reduced hypothesis space with fewer parameters, the sparse network $\hat{\theta}$ may induce a large bias, making the learning targets unreliable, as evidenced in [Sokar et al., 2022].

Moreover, we establish Theorem 3.1 to characterize the upper bound of the expected multi-step TD error under sparse models, where the multi-step return at $(s_t, \boldsymbol{u}_t)$ is $\mathcal{T}_n(s_t, \boldsymbol{u}_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_{\boldsymbol{u}} Q_{\text{tot}}(s_{t+n}, \boldsymbol{u}; \hat{\theta})$ under sparse models. As Eq (2) shows, the expected multi-step TD error comes from two parts, intrinsical policy inconsistency error and network fitting error. Thus, Eq (2) implies that the expected TD error will be enlarged if the network is sparsified improperly with a larger network fitting error. Indeed, the upper bound of the expected TD error will be enlarged if the network fitting error is increased. Subsequently, it is infeasible for the model to learn a good policy. Eq. (2) also shows that introducing a multi-step return target discounts the network fitting error by a factor of γ^n in the upper bound of the expected TD error. Thus, employing a multi-step return $\mathcal{T}_t^{(n)}$ with a sufficiently large n, or even Monte Carlo methods [Sutton and Barto, 2018], can effectively diminish the TD error caused by network sparsification for $\gamma < 1$.

Theorem 3.1. *Denote π as the target policy at timestep t, and ρ as the behavior policy generating the transitions $(s_t, \boldsymbol{u}_t, \ldots, s_{t+n}, \boldsymbol{u}_{t+n})$. Denote the network fitting error as $\epsilon(s, \boldsymbol{u}) = |Q_{tot}(s, \boldsymbol{u}; \hat{\theta}) -$*

$Q^{\pi}_{tot}(s, \boldsymbol{u})|$. *Then, the expected error between the multi-step TD target* $\mathcal{T}_n$ *conditioned on transitions from the behavior policy* ρ *and the true joint action-value function* Q^{π}_{tot} *is*

$$|\mathbb{E}_\rho[\mathcal{T}_n(s_t, \boldsymbol{u}_t)] - Q^{\pi}_{tot}(s_t, \boldsymbol{u}_t)| \leq \gamma^n \mathbb{E}_\rho[\underbrace{2\epsilon(s_{t+n}, \rho(s_{t+n})) + \epsilon(s_{t+n}, \pi(s_{t+n}))}_{\textit{Network fitting error}}] + \underbrace{|Q^{\rho}_{tot}(s_t, \boldsymbol{u}_t) - Q^{\pi}_{tot}(s_t, \boldsymbol{u}_t)|}_{\textit{Policy inconsistency error}} + \underbrace{\gamma^n \mathbb{E}_\rho[|Q^{\pi}_{tot}(s_{t+n}, \pi(s_{t+n})) - Q^{\rho}_{tot}(s_{t+n}, \rho(s_{t+n}))|]}_{\textit{Discounted policy inconsistency error}}. \tag{2}$$

Proof. Please refer to Appendix A.3. □

However, the Monte Carlo method is prone to high variance, suggesting that an optimal TD target in sparse models should be a multi-step return with a judiciously chosen step length, balancing network fitting error due to sparsification and training variance. Figure 4 illustrates model performance across different step lengths and model sizes, revealing that an optimal step length exists for various model sizes. Moreover, the optimal step length increases as model size decreases, which aligns with Theorem 3.1 due to the increased network fitting error in models with higher sparsity. The above facts suggest the need to increase the step length in the learning targets to maintain the performance of sparse models. However, the optimal step length varies across different settings. To address this, we introduce the TD(λ) target [Sutton and Barto, 2018] to achieve a good trade-off: $\mathcal{T}_\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\mathcal{T}_n$ for $\lambda \in [0, 1]$. This target averages all possible multi-step returns $\{\mathcal{T}_n\}_{n=1}^{\infty}$ into a single return using exponentially decaying weights, providing a computationally efficient approach with episode-form data. In Figure 4, we also plot two representative performances of TD(λ) under 10% and 5% model sizes, which are both close to the optimal step length under different model sizes.

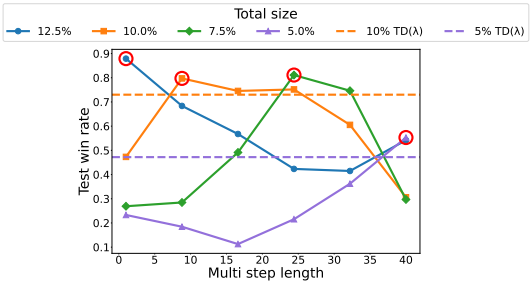


Figure 4: Performances of different step lengths.

Previous studies [Fedus et al., 2020] have highlighted that an immediate shift to multi-step targets can exacerbate policy inconsistency error as shown in Eq. (2). Since the TD(λ) target $\mathcal{T}_\lambda$ averages all potential multi-step returns $\{\mathcal{T}_n\}_{n=1}^{\infty}$, an immediate transition to this target may encounter similar issues. To address this challenge, we adopt a hybrid strategy inspired by the delayed mechanism proposed in [Tan et al., 2022]. Initially, when the training step is less than a threshold T_0, we employ one-step TD targets ($\mathcal{T}_1$) to minimize policy inconsistency errors. As training progresses and the policy stabilizes, we transit to TD(λ) targets to mitigate sparse network fitting errors. This mechanism ensures consistent and reliable learning targets throughout the sparse training process.

Soft Mellowmax Operator. The max operator in the Bellman operator poses a well-known theoretical challenge, namely overestimation, which hinders the convergence of various linear and non-linear approximation schemes [Tsitsiklis and Van Roy, 1996]. Deep MARL algorithms, including QMIX [Rashid et al., 2020b], also grapple with the overestimation problem. Several works have addressed this issue in dense MARL algorithms, such as double critics [Ackermann et al., 2019], weighted critic updates [Sarkar and Kalita, 2021], the Softmax operator [Pan et al., 2021], and the Sub-Avg operator [Wu et al., 2022]. However, these methods introduce additional computational costs, sometimes even doubling the computational budget, which is infeasible for our sparse training framework.

We turn our attention to the Soft Mellowmax operator, which has been proven effective in reducing overestimation for dense MARL algorithms in [Gan et al., 2021]. For MARL algorithms satisfying the IGM property in Eq. (1), we replace the max operator in Q_i with the Soft Mellowmax operator in Eq. (3) to mitigate overestimation bias in the joint-action Q function within sparse models:

$$\mathrm{sm}_\omega(Q_i(\tau, \cdot)) = \frac{1}{\omega} \log \left[\sum_{u \in \mathcal{U}} \frac{\exp\left(\alpha Q_i\left(\tau, u\right)\right)}{\sum_{u' \in \mathcal{U}} \exp\left(\alpha Q_i\left(\tau, u'\right)\right)} \exp\left(\omega Q_i\left(\tau, u\right)\right) \right], \tag{3}$$

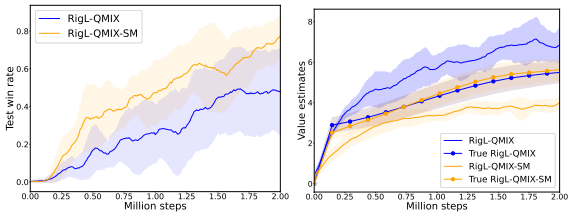
where $\omega > 0$, and $\alpha \in \mathbb{R}$. Let $\mathcal{T}$ be the value estimation operator that estimates the value of the next state s'. Theorem 3.2 shows that the Soft Mellowmax operator can reduce the severe overestimation bias in sparse models.

Theorem 3.2. *Let* $B(\mathcal{T}) = \mathbb{E}\left[\mathcal{T}\left(s'\right)\right] - \max_{\boldsymbol{u}'} Q^*_{tot}\left(s', \boldsymbol{u}'\right)$ *be the bias of value estimates of* $\mathcal{T}$. *For an arbitrary joint-action Q-function* Q_{tot}, *if there exists some* $V^*_{tot}\left(s'\right)$ *such that*

$V^*_{tot}(s') = Q^*_{tot}(s', \boldsymbol{u}')$ *for different joint actions,* $\sum_{\boldsymbol{u}'} \left(\bar{Q}_{tot}(s', \boldsymbol{u}') - V^*_{tot}(s')\right) = 0$, *and* $\frac{1}{|\boldsymbol{U}|} \sum_{\boldsymbol{u}'} \left(\bar{Q}_{tot}(s', \boldsymbol{u}') - V^*_{tot}(s')\right)^2 = C\ (C > 0)$, *then* $B(\mathcal{T}_{\text{RigL-QMIX-SM}}) \leq B(\mathcal{T}_{\text{RigL-QMIX}})$.

Proof. Please refer to Appendix A.4. □

Our empirical investigations reveal that overestimation remains a significant performance barrier in sparse models, resulting in substantial performance degradation. Figure 12 illustrates the win rates and estimated values of QMIX with and without our Soft Mellowmax operator on `3s5z` in the SMAC. Figure 5(a) shows that the performance of RigL-QMIX-SM outperforms RigL-QMIX, while Figure 5(b) demonstrates that the Soft Mellowmax operator effectively mitigates overestimation bias. These findings highlight that QMIX still faces overestimation issues in sparse models and underscore the efficacy of the Soft Mellowmax operator in addressing this problem.



(a) Win rates (b) Estimated values

Figure 5: Effects of Soft Mellowmax operator.

Additionally, the Soft Mellowmax operator introduces negligible extra computational costs, as it averages the Q function over each agent's individual action spaces rather than the joint action spaces used in the Softmax operator [Pan et al., 2021], which grow exponentially with the number of agents.

3.2 Improving the Rationality of Sample Distribution

Although we have improved the reliability of the learning target's confidence as discussed above, the learning process can still suffer from instability due to improper sparsification. This suggests the need to enhance the distribution of training samples to stabilize the training process. Improper sparsification can lead to irreversible damage to the learning path [Igl et al., 2020] and exacerbate training instability as agents learn through bootstrapping.

Generally, sparse models are more challenging to train compared to dense models due to the reduced hypothesis space [Evci et al., 2019]. Therefore, it is important to prioritize more recent training samples to estimate the true value function within the same training budget. Failing to do so can result in excessively large policy inconsistency errors, thereby damaging the learning process irreversibly.

MAST introduces a dual buffer mechanism utilizing two First-in-First-Out (FIFO) replay buffers: $\mathcal{B}_1$ (with large capacity) and $\mathcal{B}_2$ (with small capacity), with some data overlap between them. While $\mathcal{B}_1$ adopts an off-policy style, $\mathcal{B}_2$ follows an on-policy approach. During each training step, MAST samples b_1 episodes from $\mathcal{B}_1$ and b_2 episodes from $\mathcal{B}_2$, conducting a gradient update using a combined batch size of $(b_1 + b_2)$. For instance, in our experiments, we set $|\mathcal{B}_1| : |\mathcal{B}_2| = 50 : 1$ and $b_1 : b_2 = 3 : 1$. Generally, training with online data enhances learning stability, as the behavior policy closely matches the target policy. Conversely, training with offline data improves sample efficiency but can lead to instability.

Figure 6 illustrates the training dynamics for RigL-QMIX in the SMAC's `3s5z` task. The green curve with large variances highlights the training instability of QMIX in sparse models. However, with the integration of dual buffers, QMIX's training stability and efficiency are significantly improved under sparse conditions, leading to consistent policy enhancements and higher rewards. Notably, the dual buffer mechanism does not enhance dense training, suggesting that this approach is particularly effective in sparse scenarios where network parameters are crucial for ensuring stable policy improvements. Although prior works [Schaul et al., 2015, Hou et al., 2017, Banerjee et al., 2022] have explored prioritized or dynamic-capacity buffers, their applicability in this context may be limited due to the data being in episode form in value-based deep MARL algorithms, making it difficult to determine the priority of each training episode. Similarly, the dynamic buffer approach in [Tan et al., 2022] is also inapplicable, as the policy distance measure cannot be established for episode-form data. This further emphasizes the unique effectiveness of the dual buffer approach in enhancing training stability for sparse MARL models.

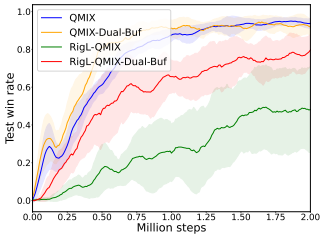


Figure 6: Different buffers.

We highlight that the performance improvement observed with MAST is not due to providing more training data samples to agents. Instead, it results from a more balanced training data distribution enabled by the utilization of dual buffers. The primary objective of incorporating dual buffers is to shift the overall distribution of training data. Figure 7 illustrates the distribution of samples induced by different policies, where behavior policy and target policy are defined in Theorem 3.1. Specifically, samples in the original single buffer are subject to the distribution of the behavior policy (blue), which introduces a policy inconsistency error d_1. However, by utilizing dual buffers, the distribution of training samples shifts towards the target policy, as there are more recent samples from the on-policy buffer. This reduces the policy inconsistency in our dual buffers, thereby bolstering the stability and effectiveness of the learning process under sparse models. Additionally, the extra on-policy buffer does not significantly reduce sample efficiency, as its capacity is small.

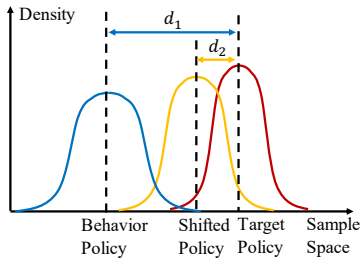


Figure 7: Distribution Shift: d_1 and d_2 are distribution distances.

4 Experiments

In this section, we conduct a comprehensive performance evaluation of MAST across various tasks in the StarCraft Multi-Agent Challenge (SMAC) [Samvelyan et al., 2019] benchmark. Additional experiments on the multi-agent MuJoCo (MAMuJoCo) [Peng et al., 2021] benchmark are provided in Appendix B.9. MAST serves as a versatile sparse training framework specifically tailored for value decomposition-based MARL algorithms. In Section 4.1, we integrate MAST with state-of-the-art value-based deep MARL algorithms, including QMIX [Rashid et al., 2020b], WQMIX [Rashid et al., 2020a], and RES [Pan et al., 2021]. We also apply MAST to a hybrid value-based and policy-based algorithm, FACMAC [Peng et al., 2021]. Subsequently, we assess the performance of sparse models generated by MAST in Section 4.2. Furthermore, a comprehensive ablation study of MAST components is detailed in Appendix B.7. Each reported result represents the average performance over eight independent runs, each utilizing distinct random seeds.

Table 1: Comparisons of MAST with different sparse training baselines: "Sp." stands for "sparsity", "Total Size" means total model parameters, and the data is all normalized w.r.t. the dense model.

Alg.	Env.	Sp.	Total Size	FLOPs (Train)	FLOPs (Test)	Tiny (%)	SS (%)	SET (%)	RigL (%)	RLx2 (%)	MAST (%)
Q-MIX	3m	95%	0.066x	0.051x	0.050x	98.3	91.6	96.0	95.3	12.1	**100.9**
	2s3z	95%	0.062x	0.051x	0.050x	83.7	73.0	77.6	69.4	45.8	**98.0**
	3s5z	90%	0.109x	0.101x	0.100x	68.2	34.0	52.3	45.2	50.1	**99.0**
	64*	90%	0.106x	0.100x	0.100x	58.2	40.2	67.1	48.7	9.9	**97.6**
	Avg.	92%	0.086x	0.076x	0.075x	77.1	59.7	73.2	64.6	29.8	**98.9**
WQ-MIX	3m	90%	0.108x	0.100x	0.100x	98.3	96.9	97.8	97.8	98.0	**98.6**
	2s3z	90%	0.106x	0.100x	0.100x	89.6	75.4	85.9	86.8	87.3	**100.2**
	3s5z	90%	0.105x	0.100x	0.100x	70.7	62.5	56.0	50.4	60.7	**96.1**
	64*	90%	0.104x	0.100x	0.100x	51.0	29.6	44.1	41.0	52.8	**98.4**
	Avg.	90%	0.106x	0.100x	0.100x	77.4	66.1	70.9	69.0	74.7	**98.1**
RES	3m	95%	0.066x	0.055x	0.050x	97.8	95.6	97.3	91.1	97.9	**99.8**
	2s3z	90%	0.111x	0.104x	0.100x	96.5	92.8	92.8	94.7	94.0	**98.4**
	3s5z	85%	0.158x	0.154x	0.150x	95.1	89.0	90.3	92.8	86.2	**99.4**
	64*	85%	0.155x	0.151x	0.150x	83.3	39.1	44.1	35.3	72.7	**104.9**
	Avg.	89%	0.122x	0.116x	0.112x	93.2	79.1	81.1	78.5	87.7	**100.6**

4.1 Comparative Evaluation

Table 1 presents a comprehensive summary of our comparative evaluation in the SMAC benchmark, where MAST is benchmarked against the following baseline methods: (i) Tiny: Utilizing tiny dense networks with a parameter count matching that of the sparse model during training. (ii) SS: Employing static sparse networks with random initialization. (iii) SET [Mocanu et al., 2018]: Pruning connections based on their magnitude and randomly expanding connections. (iv) RigL [Evci et al., 2020]: This approach leverages dynamic sparse training, akin to MAST, by removing and adding

connections based on magnitude and gradient criteria, respectively. (v) RLx2 [Tan et al., 2022]: A specialized dynamic sparse training framework tailored for single-agent reinforcement learning.

We set the same sparsity levels for both the joint Q function Q_{tot}, and each individual agent's Q function Q_i. For every algorithm and task, the sparsity level indicated in Table 1 corresponds to the highest admissible sparsity threshold of MAST. Within this range, MAST's performance consistently remains within a 3% margin compared to the dense counterpart, effectively representing the minimal sparse model size capable of achieving performance parity with the original dense model. All other baselines are evaluated under the same sparsity level as MAST. We assess the performance of each algorithm by computing the average win rate per episode over the final 20 policy evaluations conducted during training, with policy evaluations taking place at 10000-step intervals. Identical hyperparameters are employed across all scenarios.

Performance Table 1 unequivocally illustrates MAST's substantial performance superiority over all baseline methods in all four environments across the three algorithms. Notably, static sparse (SS) consistently exhibits the lowest performance on average, highlighting the difficulty of finding optimal sparse network topologies in the context of sparse MARL models. Dynamic sparse training methods, namely SET and RigL, slightly outperform SS, although their performance remains unsatisfactory. Sparse networks also, on average, underperform tiny dense networks. However, MAST significantly outpaces all other baselines, indicating the successful realization of accurate value estimation through our MAST method, which effectively guides gradient-based topology evolution. Notably, the single-agent method RLx2 consistently delivers subpar results in all experiments, potentially due to the sensitivity of the step length in the multi-step targets, and the failure of dynamic buffer for episode-form training samples.

To further substantiate the efficacy of MAST, we conduct performance comparisons across various sparsity levels in `3s5z`, as depicted in Figure 8. This reveals an intriguing observation: the performance of sparse models experiences a sharp decline beyond a critical sparsity threshold. Compared to conventional DST techniques, MAST significantly extends this critical sparsity threshold, enabling higher levels of sparsity while maintaining performance. Moreover, MAST achieves a higher critical sparsity threshold than the other two algorithms with existing baselines, e.g., SET and RigL, achieving a sparsity level of over 80% on average. However, it is essential to note that the Softmax operator in RES averages the Q function over joint action spaces, which grow exponentially with the number of agents, resulting in significantly higher computational FLOPs and making it computationally incomparable to MAST. The detailed FLOPs calculation is deferred to Appendix B.4.2.

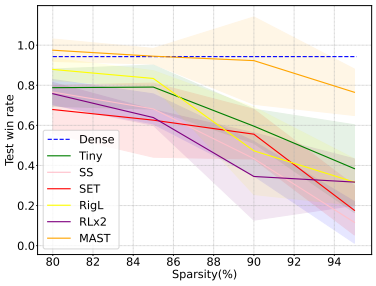
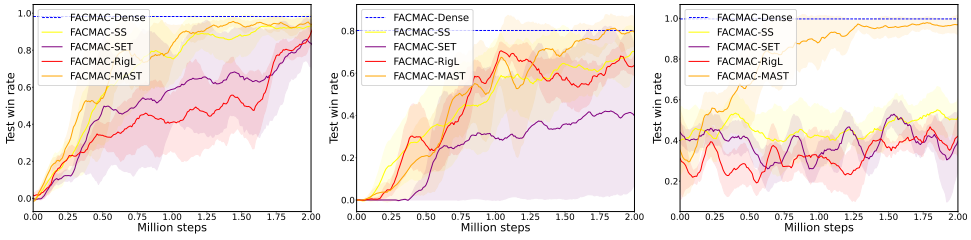


Figure 8: Different sparsity.

FLOP Reduction and Model Compression In contrast to knowledge distillation or behavior cloning methodologies, exemplified by works such as [Livne and Cohen, 2020, Vischer et al., 2022], MAST maintains a sparse network consistently throughout the entire training regimen. Consequently, MAST endows itself with a unique advantage, manifesting in a remarkable acceleration of training FLOPs. We observed an acceleration of up to $20\times$ in training and inference FLOPs for MAST-QMIX in the `2s3z` task, with an average acceleration of $10\times$, $9\times$, and $8\times$ for QMIX, WQMIX, and RES-QMIX, respectively. Moreover, MAST showcases significant model compression ratios, achieving reductions in model size ranging from $5\times$ to $20\times$ for QMIX, WQMIX, and RES-QMIX, while incurring only minor performance degradation, all below 3%.



(a) MMM (b) 2c_vs_64zg (c) bane_vs_bane

Figure 9: Mean episode win rates on different SMAC tasks with MAST-FACMAC.

Results on FACMAC In addition to pure value-based deep MARL algorithms, we also evaluate MAST with a hybrid value-based and policy-based algorithm, FACMAC [Peng et al., 2021], in SMAC. The results are presented in Figure 18. From the figure, we observe that MAST consistently achieves a performance comparable with that of the dense models, and outperforms other methods in three environments, demonstrating its applicability across different algorithms.

4.2 Sparse Models Obtained by MAST

We conduct a comparative analysis of diverse sparse network architectures. With identical sparsity levels, distinct sparse architectures lead to different hypothesis spaces. As emphasized in [Frankle and Carbin, 2019], specific architectures, such as the "winning ticket," outperform randomly generated counterparts. We compare three architectures: the "random ticket" (randomly sampled topology held constant during training), the "winning ticket" (topology from a MAST or RigL run and kept unchanged during training), and the "cheating ticket" (trained by MAST).

Figure 10 illustrates that both the "cheating ticket" and "winning ticket" by MAST achieve the highest performance, closely approaching the original dense model's performance. Importantly, using a fixed random topology during training fails to fully exploit the benefits of high sparsity, resulting in significant performance degradation. Furthermore, RigL's "winning ticket" fares poorly, akin to the "random ticket." These results underscore the advantages of our MAST approach, which automatically discovers effective sparse architectures through gradient-based topology evolution, without the need for pretraining methods, e.g., knowledge distillation [Schmitt et al., 2018]. Crucially, our MAST method incorporates key elements: the hybrid TD(λ) mechanism, Soft Mellowmax operator, and dual buffers. Compared to RigL, these components significantly improve value estimation and training stability in sparse models facilitating efficient topology evolution.

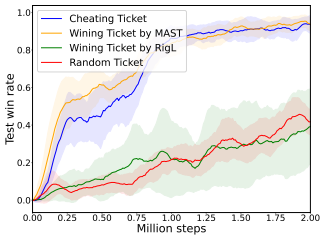


Figure 10: Comparison of different sparse masks.

Figure 11 showcases the evolving sparse mask of a hidden layer during MAST-QMIX training in 3s5z, capturing snapshots at 0, 5, 10, and 20 million steps. In Figure 11, the light pixel in row i and column j indicates the existence of the connection for input dimension j and output dimension i, while the dark pixel represents the empty connection. Notably, a pronounced shift in the mask is evident at the start of training, followed by a gradual convergence of connections within the layer onto a subset of input neurons. This convergence is discernible from the clustering of light pixels forming continuous rows in the lower segment of the final mask visualization, where several output dimensions exhibit minimal or no connections. This observation underscores the distinct roles played by various neurons in the representation process, showing the prevalent redundancy in dense models.

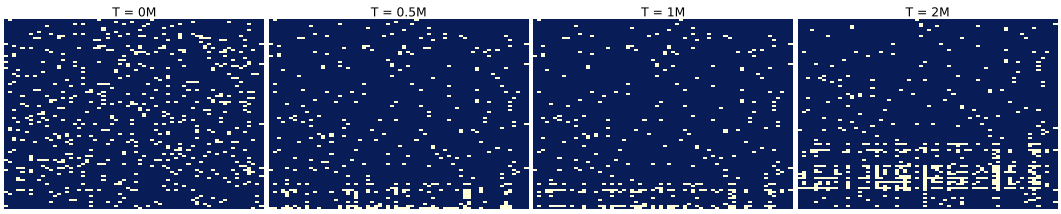
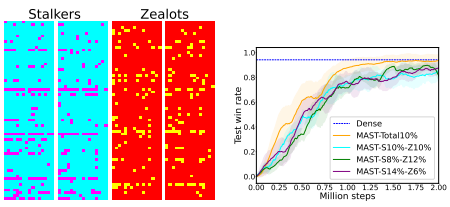


Figure 11: Visualization of weight masks in the first hidden layer of agent 1 by MAST-QMIX.

When sparsifying agent networks, MAST only requires setting the total sparsity. The sparsity of different agents is determined automatically by MAST, by concatenating all the agent networks together in our implementation based on PyMARL [Samvelyan et al., 2019] (detailed in Appendix B.3), and treating these networks as a single network during topology evolution with only a total sparsity requirement. We visualize the trained masks of different agents in 3s5z in Figure 12(a), including the masks of two stalkers and two zealots, respectively.



(a) Agent mask visualization. (b) Different sparsity patterns.

Figure 12: Agent roles.

Interestingly, we find that the network topology in the same type of agents looks very similar. However, stalkers have more connections than zealots, which aligns with the fact that stalkers play more critical roles due to their higher attack power and wider attack ranges. This observation highlights an advantage of the MAST framework, i.e., it can automatically discover the proper sparsity levels for different agents to meet the total sparsity budget. To further validate this point, we compare the adaptive allocation scheme with fixed manually-set patterns in Figure 12(b). The manual patterns include (stalker-10%, zealot-10%), (stalker-8%, zealot-12%), and (stalker-14%, zealot-6%). The results also show that the adaptive sparsity allocation in MAST outperforms other manual sparsity patterns, demonstrating the superiority of MAST.

5 Conclusion

This paper introduces MAST, a novel sparse training framework for valued-based deep MARL. We identify and address value estimation errors and policy inconsistency caused by sparsification, two significant challenges in training sparse agents. MAST offers innovative solutions: a hybrid TD(λ) target mechanism combined with the Soft Mellowmax operator for precise value estimation under extreme sparsity, and a dual buffer mechanism to reduce policy inconsistency and enhance training stability. Extensive experiments validate MAST's effectiveness in sparse training, achieving model compression ratios of $5\times$ to $20\times$ with minimal performance degradation and up to a remarkable $20\times$ reduction in FLOPs for both training and inference.

Acknowledgement

This work was supported by the Technology and Innovation Major Project of the Ministry of Science and Technology of China under Grant 2020AAA0108400 and 2020AAA0108403.

References

Johannes Ackermann, Volker Gabler, Takayuki Osa, and Masashi Sugiyama. Reducing overestimation bias in multi-agent domains using double centralized critics. *arXiv preprint arXiv:1910.01465*, 2019.

SV Albrecht, F Christianos, and L Schäfer. Multi-agent reinforcement learning: Foundations and modern approaches. *Massachusetts Institute of Technology: Cambridge, MA, USA*, 2023.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2020.

Chayan Banerjee, Zhiyong Chen, and Nasimul Noman. Improved soft actor-critic: Mixing prioritized off-policy samples with on-policy experiences. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Christopher Brix, Parnia Bahar, and Hermann Ney. Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3909–3915, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020a.

Yu-Jia Chen, Deng-Kai Chang, and Cheng Zhang. Autonomous tracking using a swarm of uavs: A constrained multi-agent reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 69(11):13702–13717, 2020b.

Filippos Christianos, Georgios Papoudakis, Muhammad A Rahman, and Stefano V Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *International Conference on Machine Learning*, pages 1989–1998. PMLR, 2021.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Arnau Colom. Empirical analysis of exploration strategies in qmix. 2021.

Felipe Leno Da Silva, Ruben Glatt, and Anna Helena Reali Costa. Simultaneously learning and advising in multiagent reinforcement learning. In *Proceedings of the 16th conference on autonomous agents and multiagent systems*, pages 1100–1108, 2017.

Shrey Desai, Hongyuan Zhan, and Ahmed Aly. Evaluating lottery tickets under distributional shifts. *EMNLP-IJCNLP 2019*, page 153, 2019.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. In *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International conference on learning representations*, 2019.

Yaozhong Gan, Zhe Zhang, and Xiaoyang Tan. Stabilizing q learning via soft mellowmax operator. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7501–7509, 2021.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pages 7766–7792. PMLR, 2022.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International conference on learning representations*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *2017 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 316–321. IEEE, 2017.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Woojun Kim and Youngchul Sung. Parameter sharing with network pruning for scalable multi-agent deep reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 1942–1950, 2023.

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

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International conference on learning representations*, 2019.

Chenghao Li, Tonghan Wang, Chengjie Wu, Qianchuan Zhao, Jun Yang, and Chongjie Zhang. Celebrating diversity in shared multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3991–4002, 2021.

Dor Livne and Kobi Cohen. Pops: Policy pruning and shrinking for deep reinforcement learning. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):789–801, 2020.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International conference on learning representations*, 2018.

Michael Mathieu, Sherjil Ozair, Srivatsan Srinivasan, Caglar Gulcehre, Shangtong Zhang, Ray Jiang, Tom Le Paine, Konrad Zolna, Richard Powell, Julian Schrittwieser, et al. Starcraft ii unplugged: Large scale offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655. PMLR, 2019.

Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Ling Pan, Tabish Rashid, Bei Peng, Longbo Huang, and Shimon Whiteson. Regularized softmax deep multi-agent q-learning. *Advances in Neural Information Processing Systems*, 34:1365–1377, 2021.

Ling Pan, Longbo Huang, Tengyu Ma, and Huazhe Xu. Plan better amid conservatism: Offline multi-agent reinforcement learning with actor rectification. In *International conference on machine learning*, pages 17221–17237. PMLR, 2022.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

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

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020b.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Tamal Sarkar and Shobhanjana Kalita. A weighted critic update approach to multi agent twin delayed deep deterministic algorithm. In *2021 IEEE 18th India Council International Conference (INDICON)*, pages 1–6. IEEE, 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

Jonathan Schwarz, Siddhant Jayakumar, Razvan Pascanu, Peter E Latham, and Yee Teh. Powerpropagation: A sparsity inducing weight reparameterisation. *Advances in neural information processing systems*, 34:28889–28903, 2021.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. 2022.

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

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 138–145, 2017.

Chuangchuang Sun, Macheng Shen, and Jonathan P How. Scaling up multiagent reinforcement learning for robotic systems: Learn an adaptive sparse communication graph. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11755–11762. IEEE, 2020.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. Rlx2: Training a sparse deep reinforcement learning model from scratch. In *International Conference on Learning Representations*, 2022.

Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38 (3):58–68, 1995.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

JN Tsitsiklis and B Van Roy. An analysis of temporal-difference learning with function approximationtechnical. *Rep. LIDS-P-2322). Lab. Inf. Decis. Syst. Massachusetts Inst. Technol. Tech. Rep*, 1996.

Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. In *International conference on learning representations*, 2022.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

Xijun Wang, Meina Kan, Shiguang Shan, and Xilin Chen. Fully learnable group convolution for acceleration of deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9049–9058, 2019.

Haolin Wu, Jianwei Zhang, Zhuang Wang, Yi Lin, and Hui Li. Sub-avg: Overestimation reduction for cooperative multi-agent reinforcement learning. *Neurocomputing*, 474:94–106, 2022.

Je Yang, JaeUk Kim, and Joo-Young Kim. Learninggroup: A real-time sparse training on fpga via learnable weight grouping for multi-agent reinforcement learning. In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–9. IEEE, 2022.

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

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International conference on learning representations*, 2020.

Hongjie Zhang, Zhuocheng He, and Jing Li. Accelerating the deep reinforcement learning with neural network compression. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

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

Supplementary Materials

A Additional Details for MAST Framework **16**

A.1 Comprehensive Related Work 16

A.2 Decentralized Partially Observable Markov Decision Process 17

A.3 Proof of Theorem 3.1 17

A.4 Proof of Theorem 3.2 18

A.5 MAST with Different Algorithms 18

A.6 Limitations of MAST 21

B Experimental Details **21**

B.1 Hardware Setup 21

B.2 Environment 21

B.3 Hyperparameter Settings 22

B.4 Calculation of Model Sizes and FLOPs 22

B.5 Training Curves of Comparative Evaluation in Section 4.1 26

B.6 Standard Deviations of Results in Table 1 27

B.7 Ablation Study 28

B.8 Sensitivity Analysis for Hyperparameters 29

B.9 Experiments on QMIX in Multi-Agent MuJoCo 30

B.10 Experiments on FACMAC in SMAC 30

B.11 Visualization of Sparse Masks 30

A Additional Details for MAST Framework

A.1 Comprehensive Related Work

Sparse networks, initially proposed in deep supervised learning can train a 90%-sparse network without performance degradation from scratch. However, for deep reinforcement learning, the learning target is not fixed but evolves in a bootstrap way [Tesauro et al., 1995], and the distribution of the training data can also be non-stationary [Desai et al., 2019], which makes the sparse training more difficult. In the following, we list some representative works for training sparse models from supervised learning to reinforcement learning.

Sparse Models in Supervised Learning Various techniques have been explored for creating sparse networks, ranging from pruning pre-trained dense networks [Han et al., 2015, 2016, Srinivas et al., 2017], to employing methods like derivatives [Dong et al., 2017, Molchanov et al., 2019], regularization [Louizos et al., 2018], dropout [Molchanov et al., 2017], and weight reparameterization [Schwarz et al., 2021]. Another avenue of research revolves around the Lottery Ticket Hypothesis (LTH) [Frankle and Carbin, 2019], which posits the feasibility of training sparse networks from scratch, provided a sparse “winning ticket" initialization is identified. This hypothesis has garnered support in various deep learning models [Chen et al., 2020a, Brix et al., 2020]. Additionally, there is a body of work dedicated to training sparse neural networks from the outset, involving techniques that evolve the structures of sparse networks during training. Examples include Deep Rewiring (DeepR) [Bellec et al., 2018], Sparse Evolutionary Training (SET) [Mocanu et al., 2018], Dynamic Sparse Reparameterization (DSR) [Mostafa and Wang, 2019], Sparse Networks from Scratch (SNFS) [Dettmers and Zettlemoyer, 2019], and Rigged Lottery (RigL) [Evci et al., 2020]. Furthermore,

methods like Single-Shot Network Pruning (SNIP) [Lee et al., 2019] and Gradient Signal Preservation (GraSP) [Wang et al., 2020] are geared towards identifying static sparse networks prior to training.

Sparse Models in Single-Agent RL Existing research [Schmitt et al., 2018, Zhang et al., 2019] has employed knowledge distillation with static data to ensure training stability and generate small dense agents. Policy Pruning and Shrinking (PoPs) [Livne and Cohen, 2020] generates sparse agents through iterative policy pruning, while the LTH in DRL is first indentified in [Yu et al., 2020]. Another line of investigation aims to train sparse DRL models from scratch, eliminating the necessity of pre-training a dense teacher. Specifically, [Sokar et al., 2022] introduces the Sparse Evolutionary Training (SET) approach, achieving a remarkable 50% sparsity level through topology evolution in DRL. Additionally, [Graesser et al., 2022] observes that pruning often yields superior results, with plain dynamic sparse training methods, including SET and RigL, significantly outperforming static sparse training approaches. More recently, RLx2 [Tan et al., 2022] has demonstrated the capacity to train DRL agents with highly sparse neural networks from scratch. Nevertheless, the application of RLx2 in MARL yields poor results, as demonstrated in Section 4.1.

Sparse Models in MARL Existing works have made attempts to train sparse MARL agents, such as [Yang et al., 2022], which prunes networks for multiple agents during training, employing weight grouping [Wang et al., 2019]. Another avenue of sparse MARL research seeks to enhance the scalability of MARL algorithms through sparse architectural modifications. For instance, [Sun et al., 2020] proposes the use of a sparse communication graph with graph neural networks to reduce problem scale. [Kim and Sung, 2023] adopts structured pruning for a deep neural network to extend the scalability. Yet another strand of sparse MARL focuses on parameter sharing between agents to reduce the number of trainable parameters, with representative works including [Gupta et al., 2017, Li et al., 2021, Christianos et al., 2021]. However, existing methods fail to maintain high sparsity throughout the training process, such that the FLOPs reduction during training is incomparable to the MAST framework outlined in our paper.

A.2 Decentralized Partially Observable Markov Decision Process

We model the MARL problem as a decentralized partially observable Markov decision process (Dec-POMDP) [Oliehoek et al., 2016], represented by a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, \gamma \rangle$, where $\mathcal{N} = \{1, \ldots, N\}$ denotes the finite set of agents, $\mathcal{S}$ is the global state space, $\mathcal{U}$ is the action space for an agent, P is the transition probability, r is the reward function, $\mathcal{Z}$ is the observation space for an agent, O is the observation function, and and $\gamma \in [0, 1)$ is the discount factor. At each timestep t, each agent $i \in \mathcal{N}$ receives an observation $z \in \mathcal{Z}$ from the observation function $O(s, i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{Z}$ due to partial observability, and chooses an action $u_i \in \mathcal{U}$, which forms a joint action $\boldsymbol{u} \in \boldsymbol{\mathcal{U}} \equiv \mathcal{U}^n$. The joint action $\boldsymbol{u}$ taken by all agents leads to a transition to the next state s' according to transition probability $P(s' \mid s, \boldsymbol{u}) : \mathcal{S} \times \boldsymbol{\mathcal{U}} \times \mathcal{S} \mapsto [0, 1]$ and a joint reward $r(s, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U} \mapsto \mathbb{R}$. As the time goes by, each agent $i \in \mathcal{N}$ has an action-observation history $\tau_i \in \boldsymbol{\mathcal{T}} \equiv (\mathcal{Z} \times \mathcal{U})^*$, where $\boldsymbol{\mathcal{T}}$ is the history space. Based on τ_i, each agent i outputs an action u_i according to its constructed policy $\pi_i(u_i \mid \tau_i) : \boldsymbol{\mathcal{T}} \times \mathcal{U} \mapsto [0, 1]$. The goal of agents is to find an optimal joint policy $\boldsymbol{\pi} = \langle \pi_1, \ldots, \pi_N \rangle$, which maximize the joint cumulative rewards $J(s_0; \boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{u}_t \sim \boldsymbol{\pi}(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, \boldsymbol{u}_t)} \left[\sum_{t=0}^{\infty} \gamma^i r(s_t, \boldsymbol{u}_t) \right]$, where s_0 is the initial state. The joint action-value function associated with policy $\boldsymbol{\pi}$ is defined as $Q^{\boldsymbol{\pi}}(s_t, \boldsymbol{u}_t) = \mathbb{E}_{\boldsymbol{u}_{t+i} \sim \boldsymbol{\pi}(\cdot|s_{t+i}), s_{t+i+1} \sim P(\cdot|s_{t+i}, \boldsymbol{u}_{t+i})} \left[\sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, \boldsymbol{u}_{t+i}) \right]$.

A.3 Proof of Theorem 3.1

Proof. By the definitions of multi-step targets, we have

$$
\begin{aligned}
&\mathbb{E}_\rho[\mathcal{T}_n(s_t, \boldsymbol{u}_t)] \\
=&\mathbb{E}_\rho[\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_{\boldsymbol{u}} Q_{\text{tot}}\left(s_{t+n}, \boldsymbol{u}; \theta\right)] \\
=&\mathbb{E}_\rho[\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n Q_{\text{tot}}\left(s_{t+n}, \pi(s_{t+n}); \theta\right)]
\end{aligned}
$$

$$
\begin{aligned}
=&\mathbb{E}_\rho[\sum_{k=0}^{n-1}\gamma^k r_{t+k}+\gamma^n Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)]\\
&+\gamma^n\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\pi(s_{t+n});\theta\right)-Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)]\\
=&\mathbb{E}_\rho[\sum_{k=0}^{n-1}\gamma^k r_{t+k}+\gamma^n Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)]\\
&+\gamma^n\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)-Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)]\\
&+\gamma^n\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\pi(s_{t+n});\theta\right)-Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)]\\
=&Q^\rho_{\text{tot}}\left(s_t,\boldsymbol{u}_t\right)+\gamma^n\mathbb{E}_\rho[\epsilon(s_{t+n},\rho(s_{t+n}))]+\gamma^n\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\pi(s_{t+n});\theta\right)-Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)].
\end{aligned}
\tag{4}
$$

Besides, we also have

$$
\begin{aligned}
&\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\pi(s_{t+n});\theta\right)-Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)]\\
=&\mathbb{E}_\rho[Q_{\text{tot}}\left(s_{t+n},\pi(s_{t+n});\theta\right)-Q^\pi_{\text{tot}}\left(s_{t+n},\pi(s_{t+n})\right)]\\
&+\mathbb{E}_\rho[Q^\pi_{\text{tot}}\left(s_{t+n},\pi(s_{t+n})\right)-Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)]\\
&+\mathbb{E}_\rho[Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)-Q_{\text{tot}}\left(s_{t+n},\rho(s_{t+n});\phi\right)]\\
=&\mathbb{E}_\rho[\epsilon(s_{t+n},\pi(s_{t+n}))]+\mathbb{E}_\rho[Q^\pi_{\text{tot}}\left(s_{t+n},\pi(s_{t+n})\right)-Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)]+\mathbb{E}_\rho[\epsilon(s_{t+n},\rho(s_{t+n}))].
\end{aligned}
\tag{5}
$$

Thus, combining Eq. (4) and Eq. (5) gives

$$
\begin{aligned}
\mathbb{E}_\rho[\mathcal{T}_n(s_t,\boldsymbol{u}_t)]-Q^\pi_{\text{tot}}\left(s_t,\boldsymbol{u}_t\right)=\gamma^n\mathbb{E}_\rho[&\underbrace{2\epsilon(s_{t+n},\rho(s_{t+n}))+\epsilon(s_{t+n},\pi(s_{t+n}))}_{\text{Network fitting error}}]\\
+\underbrace{Q^\rho_{\text{tot}}\left(s_t,\boldsymbol{u}_t\right)-Q^\pi_{\text{tot}}\left(s_t,\boldsymbol{u}_t\right)}_{\text{Policy inconsistency error}}&+\underbrace{\gamma^n\mathbb{E}_\rho[Q^\pi_{\text{tot}}\left(s_{t+n},\pi(s_{t+n})\right)-Q^\rho_{\text{tot}}\left(s_{t+n},\rho(s_{t+n})\right)]}_{\text{Discounted policy inconsistency error}}.
\end{aligned}
$$

□

A.4 Proof of Theorem 3.2

Relace the Softmax operator in the proof of Theorem 3 in [Pan et al., 2021] with Eq. (3) gives the result directly.

A.5 MAST with Different Algorithms

In this section, we present the pseudocode implementations of MAST for QMIX [Rashid et al., 2020b] and WQMIX [Rashid et al., 2020a] in Algorithm 2 and Algorithm 3, respectively. It is noteworthy that RES [Pan et al., 2021] exclusively modifies the training target without any alterations to the learning protocol or network structure. Consequently, the implementation of MAST with RES mirrors that of QMIX.

Crucially, MAST stands as a versatile sparse training framework, applicable to a range of value decomposition-based MARL algorithms, extending well beyond QMIX, WQMIX[3], and RES. Furthermore, MAST's three innovative components—hybrid TD(λ), Soft Mellowmax operator, and dual buffer—can be employed independently, depending on the specific algorithm's requirements. This flexible framework empowers the training of sparse networks from the ground up, accommodating a wide array of MARL algorithms.

In the following, we delineate the essential steps of implementing MAST with QMIX (Algorithm 2). The steps for WQMIX are nearly identical, with the exception of unrestricted agent networks and the unrestricted mixing network's inclusion. Also, note that we follow the symbol definitions from [Colom, 2021] in Algorithm 2 and 3.

[3]Note that WQMIX encompasses two distinct instantiations, namely Optimistically-Weighted (OW) QMIX and Centrally-Weighted (CW) QMIX. In this paper, we specifically focus on OWQMIX.

Gradient-based Topology Evolution: The process of topology evolution is executed within Lines 31-33 in Algorithm 2. Specifically, the topology evolution update occurs at intervals of Δ_m timesteps. For a comprehensive understanding of additional hyperparameters pertaining to topology evolution, please refer to the definitions provided in Algorithm 1.

TD Targets: Hybrid TD(λ) with Soft Mellowmax operator is computed in the Line 25 in Algorithm 2, which modify the TD target y as follows:

$$y_{\mathrm{S}} = \begin{cases} G_t^{(1)}, & \text{if } t < T_0. \\ (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \mathcal{T}_t^{(n)}, & \text{Otherwise.} \end{cases} \tag{6}$$

Algorithm 2 MAST-QMIX

1: Initialize sparse agent networks, mixing network and hypernetwork with random parameters θ and random masks M_θ with determined sparsity S .
2: $\hat{\theta} \leftarrow \theta \odot M_\theta$ *// Start with a random sparse network*
3: Initialize target networks $\hat{\theta}^- \leftarrow \hat{\theta}$
4: Set the learning rate to α
5: Initialize the replay buffer $\mathcal{B}_1 \leftarrow \{\}$ with large capacity C_1 and $\mathcal{B}_2 \leftarrow \{\}$ with small capacity C_2
6: Initialize training step $\leftarrow 0$
7: **while** step $< T_{max}$ **do**
8: $t \leftarrow 0$
9: $s_0 \leftarrow$ initial state
10: **while** $s_t \neq$ terminal and $t <$ episode limit **do**
11: **for** each agent a **do**
12: $\tau_t^a \leftarrow \tau_{t-1}^a \cup \{(o_t, u_{t-1})\}$
13: $\epsilon \leftarrow$ epsilon-schedule(step)
14: $u_t^a \leftarrow \begin{cases} \mathrm{argmax}_{u_t^a} Q(\tau_t^a, u_t^a) & \text{with probability } 1-\epsilon \\ \mathrm{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$ *// ϵ-greedy exploration*
15: **end for**
16: Get reward r_t and next state s_{t+1}
17: $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$ *// Data in the buffer is of episodes form.*
18: $\mathcal{B}_2 \leftarrow \mathcal{B}_2 \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$
19: $t \leftarrow t+1$,step $\leftarrow$ step $+1$
20: **end while**
21: **if** $|\mathcal{B}_1| >$ batch-size **then**
22: $b \leftarrow$ random batch of episodes from $\mathcal{B}_1$ and $\mathcal{B}_2$ *// Sample from dual buffers.*
23: **for** each timestep t in each episode in batch b **do**
24:

$$Q_{tot} \leftarrow \text{Mixing-network}\left((Q_1(\tau_t^1, u_t^1), \cdots, Q_n(\tau_t^n, u_t^n)); \text{Hypernetwork}(s_t; \hat{\theta})\right)$$

25: Compute TD target y according to Eq. (6). *// TD(λ) targets with Soft Mellowmax operator.*
26: **end for**
27: $\Delta Q_{tot} \leftarrow y - Q_{tot}$
28: $\Delta\hat{\theta} \leftarrow \nabla_{\hat{\theta}} \frac{1}{b} \sum (\Delta Q_{tot})^2$
29: $\hat{\theta} \leftarrow \hat{\theta} - \alpha \Delta\hat{\theta}$
30: **end if**
31: **if** step $\mod \Delta_m = 0$ **then**
32: Topology_Evolution(networks$_{\hat{\theta}}$ by Algorithm 1.
33: **end if**
34: **if** step $\mod I = 0$, where is the target network update interval **then**
35: $\hat{\theta}^- \leftarrow \hat{\theta}$ *// Update target network.*
36: $\hat{\theta}^- \leftarrow \hat{\theta}^- \odot M_{\hat{\theta}}$
37: **end if**
38: **end while**

Algorithm 3 MAST-(OW)QMIX

Initialize sparse agent networks, mixing network and hypernetwork with random parameters θ and random masks M_θ with determined sparsity S.
Initialize unrestricted agent networks and unrestricted mixing network with random parameters ϕ and random masks M_ϕ with determined sparsity S.
$\hat{\theta} \leftarrow \theta \odot M_\theta$, $\hat{\phi} \leftarrow \phi \odot M_\phi$ *// Start with a random sparse network*
Initialize target networks $\hat{\theta}^- \leftarrow \hat{\theta}$,$\hat{\phi}^- \leftarrow \hat{\phi}$
Set the learning to rate α
Initialize the replay buffer $\mathcal{B}_1 \leftarrow \{\}$ with large capacity C_1 and $\mathcal{B}_2 \leftarrow \{\}$ with small capacity C_2

Initialize training step $\leftarrow 0$
while step $< T_{max}$ **do**
  $t \leftarrow 0$,
  $s_0 \leftarrow$ initial state
  while $s_t \neq$ terminal and $t <$ episode limit **do**
    for each agent a **do**
      $\tau_t^a \leftarrow \tau_{t-1}^a \cup \{(o_t, u_{t-1})\}$
      $\epsilon \leftarrow$ epsilon-schedule(step)
      $u_t^a \leftarrow \begin{cases} \text{argmax}_{u_t^a} Q(\tau_t^a, u_t^a; \hat{\theta}) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$ *// ϵ-greedy exploration*
    end for
    Get reward r_t and next state s_{t+1}
    $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$ *// Data in the buffer is of episodes form.*
    $\mathcal{B}_2 \leftarrow \mathcal{B}_2 \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$
    $t \leftarrow t+1$, step $\leftarrow$ step $+1$
  end while
  if $|\mathcal{B}_1| >$ batch-size **then**
    $b \leftarrow$ random batch of episodes from $\mathcal{B}_1$ and $\mathcal{B}_2$ *// Sample from dual buffers.*
    for each timestep t in each episode in batch b **do**

$$Q_{tot} \leftarrow \text{Mixing-network}\left((Q_1(\tau_t^1, u_t^1; \hat{\theta}), ..., Q_n(\tau_t^n, u_t^n; \hat{\theta})); \text{Hypernetwork}(s_t; \hat{\theta})\right)$$

$$\hat{Q}^* \leftarrow \text{Unrestricted-Mixing-network}\left(Q_1(\tau_t^1, u_t^1; \hat{\phi}), ..., Q_n(\tau_t^n, u_t^n; \hat{\phi}), s_t\right)$$

      Compute TD target y with target Unrestricted-Mixing network according to Eq. (6). *// TD(λ) targets with Soft Mellowmax operator.*
      $\omega(s_t, \mathbf{u_t}) \leftarrow \begin{cases} 1, & Q_{tot} < y \\ \alpha, & \text{otherwise.} \end{cases}$
    end for
    $\Delta Q_{tot} \leftarrow y - Q_{tot}$
    $\Delta\hat{\theta} \leftarrow \nabla_{\hat{\theta}} \frac{1}{b} \sum \omega(s, \mathbf{u})(\Delta Q_{tot})^2$
    $\hat{\theta} \leftarrow \hat{\theta} - \alpha\Delta\hat{\theta}$
    $\Delta\hat{Q}^* \leftarrow y - \hat{Q}^*$
    $\Delta\hat{\phi} \leftarrow \nabla_{\hat{\phi}} \frac{1}{b} \sum (\Delta\hat{Q}^*)^2$
    $\hat{\phi} \leftarrow \hat{\phi} - \alpha\Delta\hat{\phi}$
  end if
  if step $\mod \Delta_m = 0$ **then**
    Topology_Evolution(networks$_{\hat{\theta}}$) and Topology_Evolution(networks$_{\hat{\phi}}$) by Algorithm 1.
  end if
  if step $\mod I = 0$, where is the target network update interval **then**
    $\hat{\theta}^- \leftarrow \hat{\theta}, \hat{\phi}^- \leftarrow \hat{\phi}$
    $\hat{\theta}^- \leftarrow \hat{\theta}^- \odot M_{\hat{\theta}}, \hat{\phi}^- \leftarrow \hat{\phi}^- \odot M_{\hat{\phi}}$
  end if
end while

Here, $\lambda \in [0,1]$ is a hyperparameter, and $\mathcal{T}_t^{(n)} = \sum_{i=t}^{t+n} \gamma^{i-t} r_i + \gamma^{n+1} f_s \left(\mathrm{sm}_\omega(\bar{Q}_1(\tau_1,\cdot),\ldots,\mathrm{sm}_\omega(\bar{Q}_N(\tau_N,\cdot)\right)$, where f_s denotes the mixing network and $\bar{Q}_i$ is the target network of Q_i. The loss function of MAST, $\mathcal{L}_{\mathrm{S}}(\theta)$, is defined as:

$$\mathcal{L}_{\mathrm{S}}(\theta) = \mathbb{E}_{(s,\boldsymbol{u},r,s')\sim\mathcal{B}_1\cup\mathcal{B}_2}\left[\left(y_{\mathrm{S}} - Q_{tot}(s,\boldsymbol{u})\right)^2\right] \tag{7}$$

Dual Buffers: With the creation of two buffers $\mathcal{B}_1$ and $\mathcal{B}_2$, the gradient update with data sampled from dual buffers is performed in Lines 21-30 in Algorithm 2.

A.6 Limitations of MAST

This paper introduces MAST, a novel framework for sparse training in deep MARL, leveraging gradient-based topology evolution to explore network configurations efficiently. However, understanding its limitations is crucial for guiding future research efforts.

Hyperparameters: MAST relies on multiple hyperparameters for its key components: topology evolution, TD(λ) targets with Soft Mellowmax Operator, and dual buffers. Future work could explore methods to automatically determine these hyperparameters or streamline the sparse training process with fewer tunable settings.

Implementation: While MAST achieves efficient MARL agent training with minimal performance trade-offs using ultra-sparse networks surpassing 90% sparsity, its current use of unstructured sparsity poses challenges for running acceleration. The theoretical reduction in FLOPs might not directly translate to reduced running time. Future research should aim to implement MAST in a structured sparsity pattern to bridge this gap between theoretical efficiency and practical implementation.

B Experimental Details

In this section, we offer comprehensive experimental insights, encompassing hardware configurations, environment specifications, hyperparameter settings, model size computations, FLOPs calculations, and supplementary experimental findings.

B.1 Hardware Setup

Our experiments are implemented with PyTorch 2.0.0 [Paszke et al., 2017] and run on $4\times$ NVIDIA GTX Titan X (Pascal) GPUs. Each run needs about $12 \sim 24$ hours for QMIX or WQMIX, and about $24 \sim 72$ hours for RES for two million steps. depends on the environment types.

B.2 Environment

We assess the performance of our MAST framework using the SMAC benchmark [Samvelyan et al., 2019], a dedicated platform for collaborative multi-agent reinforcement learning research based on Blizzard's StarCraft II real-time strategy game, specifically version 4.10. It is important to note that performance may vary across different versions. Our experimental evaluation encompasses four distinct maps, each of which is described in detail below.

- `3m`: An easy map, where the agents are 3 Marines, and the enemiesa are 3 Marines.
- `2s3z`: An easy map, where the agents are 2 Stalkers and 3 Zealots, and the enemies are 2 Stalkers and 3 Zealots.
- `3s5z`: An easy map, where the agents are 3 Stalkers and 5 Zealots, and the enemies are 3 Stalkers and 5 Zealots.
- `2c_vs_64zg`: A hard map, where the agents are 2 Colossi, and the enemies are 64 Zerglings.

We also evaluate MAST on the Multi-Agent MuJoCo (MAMuJoCo) benchmark from [Peng et al., 2021], which is an environment designed for evaluating continuous MARL algorithms, focusing on cooperative robotic control tasks. It extends the single-agent MuJoCo framework included with OpenAI Gym [Brockman et al., 2016]. Inspired by modular robotics, MAMuJoCo includes scenarios with a large number of agents, aiming to stimulate progress in continuous MARL by providing diverse

and challenging tasks for decentralized coordination. We test MAST-COMIX in the Humanoid, Humanoid Standup, and ManyAgent Swimmer scenarios in MAMuJoCo. The environments are tested using their default configurations, with other settings following FACMAC [Peng et al., 2021]. Specifically, we set the maximum observation distance to $k = 0$. In the ManyAgent Swimmer scenario, we configure 10 agents, each controlling a consecutive segment of length 2. The agent network architecture of MAST-COMIX uses an MLP with two hidden layers of 400 dimensions each, following the settings in FACMAC. All other hyperparameters are also based on FACMAC.

B.3 Hyperparameter Settings

Table 3 provides a comprehensive overview of the hyperparameters employed in our experiments for MAST-QMIX, MAST-WQMIX, and MAST-RES. It includes detailed specifications for network parameters, RL parameters, and topology evolution parameters, allowing for a thorough understanding of our configurations. Besides, MAST is implemented based on the PyMARL [Samvelyan et al., 2019] framework with the same network structures and hyperparameters as given in Table 3. We also provide a hyperparameter recommendation for three key components, i.e. gradient-based topology evolution, Soft Mellowmax enabled hybrid TD(λ) targets and dual buffers, in Table 2 for deployment MAST framework in other problems.

Table 2: Recommendation for Key Hyperparameters in MAST.

Category	**Hyperparameter**	**Value**
Topology Evolution	Initial mask update fraction ζ_0	0.5
	Mask update interval Δ_m	200 episodes
TD Targets	Burn-in time T_0	$3/8$ of total training steps
	λ value in TD(λ)	0.6 or 0.8
	α in soft mellow-max operator	1
	ω in soft mellow-max operator	10
Dual Buffer	Offline buffer size C_1	5×10^3 episodes
	Online buffer size C_2	128 episodes
	Sample partition of online and offline buffer	2:6

Besides, to extend the existing QMIX to continuous action spaces, we utilized COMIX from FACMAC [Peng et al., 2021] for our experiments, which employs the Cross-Entropy Method (CEM) for approximate greedy action selection. The hyperparameter configuration of CEM also follows FACMAC settings.

B.4 Calculation of Model Sizes and FLOPs

B.4.1 Model Size

First, we delineate the calculation of model sizes, which refers to the total number of parameters within the model.

- For a sparse network with L fully-connected layers, the model size, as expressed in prior works [Evci et al., 2020, Tan et al., 2022], can be computed using the equation:

$$M_{\text{linear}} = \sum_{l=1}^{L} (1 - S_l) I_l O_l, \tag{8}$$

  where S_l represents the sparsity, I_l is the input dimensionality, and O_l is the output dimensionality of the l-th layer.

- For a sparse network with L GRU layers, considering the presence of 3 gates in a single layer, the model size can be determined using the equation:

$$M_{\text{GRU}} = \sum_{l=1}^{L} (1 - S_l) \times 3 \times h_l \times (h_l + I_l), \tag{9}$$

  where h_l represents the hidden state dimensionality.

Table 3: Hyperparameters of MAST-QMIX, MAST-WQMIX and MAST-RES.

Category	**Hyperparameter**	**Value**
Shared Hyperparameters	Optimizer	RMSProp
	Learning rate α	5×10^{-4}
	Discount factor γ	0.99
	Number of hidden units per layer of agent network	64
	Hidden dimensions in the GRU layer of agent network	64
	Embedded dimensions of mixing network	32
	Hypernet layers of mixing network Embedded dimensions of hypernetwork	2 64
	Activation Function	ReLU
	Batch size B	32 episodes
	Warmup steps	50000
	Initial ϵ	1.0
	Final ϵ	0.05
	Double DQN update	True
	Target network update interval I	200 episodes
	Initial mask update fraction ζ_0	0.5
	Mask update interval Δ_m	timesteps of 200 episodes
	Offline buffer size C_1	5×10^3 episodes
	Online buffer size C_2	128 episodes
	Burn-in time T_0	7.5×10^5
	α in soft mellow-max operator	1
	ω in soft mellow-max operator	10
	Number of episodes in a sampled batch of offline buffer S_1	20
	Number of episodes in a sampled batch of online buffer S_2	12
Hyperparameters for MAST-QMIX	Linearly annealing steps for ϵ	50k
	λ value in TD(λ)	0.8
Hyperparameters for MAST-WQMIX	Linearly annealing steps for ϵ	100k
	λ value in TD(λ)	0.6
	Coefficient of Q_{tot} loss	1
	Coefficient of $\hat{Q}^*$ loss	1
	Embedded dimensions of unrestricted mixing network	256
	Embedded number of actions of unrestricted agent network	1
	α in weighting function	0.1
Hyperparameters for MAST-RES	Linearly annealing steps for ϵ	50k
	λ value in TD(λ)	0.8
	λ value in Softmax operator	0.05
	Inverse temperature β	5.0

Specifically, the "Total Size" column in Table 1 within the manuscript encompasses the model size, including both agent and mixing networks during training. For QMIX, WQMIX, and RES, target networks are employed as target agent networks and target mixing networks. We denote the model sizes of the agent network, mixing network, unrestricted agent network, and unrestricted mixing network as M_{Agent}, M_{Mix}, $M_{\text{Unrestricted-Agent}}$, and $M_{\text{Unrestricted-Mix}}$, respectively. Detailed calculations of these model sizes are provided in the second column of Table 4.

Table 4: FLOPs and model size for MAST-QMIX , MAST-WQMIX and MAST-RES.

Algorithm	Model size	Training FLOPs	Inference FLOPs
MAST-QMIX	$2M_{\text{Agent}} + 2M_{\text{Mix}}$	$4B(\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}})$	$\text{FLOPs}_{\text{Agent}}$
MAST-WQMIX	$M_{\text{Agent}} + M_{\text{Mix}} + 2M_{\text{Unrestricted-Agent}} + 2M_{\text{Unrestricted-Mix}}$	$3B(\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}}) + 4B \cdot \text{FLOPs}_{\text{Unrestricted-Agent}} + 4B \cdot \text{FLOPs}_{\text{Unrestricted-Mix}}$	$\text{FLOPs}_{\text{Agent}}$
MAST-RES	$2M_{\text{Agent}} + 2M_{\text{Mix}}$	$4B \cdot \text{FLOPs}_{\text{Agent}} + (5 + nm)B \cdot \text{FLOPs}_{\text{Mix}}$	$\text{FLOPs}_{\text{Agent}}$

B.4.2 FLOPs Calculation

Initially, for a sparse network with L fully-connected layers, the required FLOPs for a forward pass are computed as follows (also adopted in [Evci et al., 2020] and [Tan et al., 2022]):

$$\text{FLOPs} = \sum_{l=1}^{L} (1 - S_l)(2I_l - 1)O_l, \tag{10}$$

where S_l is the sparsity, I_l is the input dimensionality, and O_l is the output dimensionality of the l-th layer. Similarly, for a sparse network with L GRU [Chung et al., 2014] layers, considering the presence of 3 gates in a single layer, the required FLOPs for a forward pass are:

$$\text{FLOPs} = \sum_{l=1}^{L} (1 - S_l) \times 3 \times h_l \times [2(h_l + I_l) - 1], \tag{11}$$

where h_l is the hidden state dimensionality.

We denote B as the batch size employed in the training process, and $\text{FLOPs}_{\text{Agent}}$ and $\text{FLOPs}_{\text{Mix}}$ as the FLOPs required for a forward pass in the agent and mixing networks, respectively. The inference FLOPs correspond exactly to $\text{FLOPs}_{\text{Agent}}$, as detailed in the last column of Table 4. When it comes to training FLOPs, the calculation encompasses multiple forward and backward passes across various networks, which will be thoroughly elucidated later. Specifically, we compute the FLOPs necessary for each training iteration. Additionally, we omit the FLOPs associated with the following processes, as they exert minimal influence on the ultimate result:

- **Interaction with the environment:** This operation, where agents decide actions for interaction with the environment, incurs FLOPs equivalent to $\text{FLOPs}_{\text{Agent}}$. Notably, this value is considerably smaller than the FLOPs required for network updates, as evident in Table 4, given that $B \gg 1$.
- **Updating target networks:** Each parameter in the networks is updated as $\theta' \leftarrow \theta$. Consequently, the number of FLOPs in this step mirrors the model size, and is thus negligible.
- **Topology evolution:** This element is executed every 200 gradient updates. To be precise, the average FLOPs involved in topology evolution are computed as $B \times \frac{2\text{FLOPs}_{\text{Agent}}}{(1-S^{(a)})\Delta_m}$ for the agent, and $B \times \frac{2\text{FLOPs}_{\text{Mix}}}{(1-S^{(m)})\Delta_m}$ for the mixer. Given that $\Delta_m = 200$, the FLOPs incurred by topology evolution are negligible.

Therefore, our primary focus shifts to the FLOPs related to updating the agent and mixer. We will first delve into the details for QMIX, with similar considerations for WQMIX and RES.

B.4.3 Training FLOPs Calculation in QMIX

Recall the way to update networks in QMIX is given by

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum (y_t - Q_{tot}(s_i, a_i; \theta))^2, \tag{12}$$

where B is the batch size. Subsequently, we can compute the FLOPs of training as:

$$\text{FLOPs}_{\text{train}} = \text{FLOPs}_{\text{TD_target}} + \text{FLOPs}_{\text{compute_loss}} + \text{FLOPs}_{\text{backward_pass}}, \tag{13}$$

where $\text{FLOPs}_{\text{TD_target}}$, $\text{FLOPs}_{\text{compute_loss}}$, and $\text{FLOPs}_{\text{backward_pass}}$ refer to the numbers of FLOPs in computing the TD targets in forward pass, loss function in forward pass, and gradients in backward pass (backward-propagation), respectively. By Eq. (6) and (7), we have:

$$\begin{aligned} \text{FLOPs}_{\text{TD_target}} &= B \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}}), \\ \text{FLOPs}_{\text{compute_loss}} &= B \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}}). \end{aligned} \tag{14}$$

For the FLOPs of gradients backward propagation, $\text{FLOPs}_{\text{backward_pass}}$, we compute it as two times the computational expense of the forward pass, which is adopted in existing literature [Evci et al., 2020], i.e.,

$$\text{FLOPs}_{\text{backward_pass}} = B \times 2 \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}}), \tag{15}$$

Combining Eq. (13), Eq. (14), and Eq. (15), the FLOPs of training in QMIX is:

$$\text{FLOPs}_{\text{train}} = B \times 4 \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}}). \tag{16}$$

B.4.4 Training FLOPs Calculation in WQMIX

The way to update the networks in WQMIX is different from that in QMIX. Specifically, denote the parameters of the original network and unrestricted network as θ and ϕ, respectively, which are updated according to

$$\begin{aligned} \theta \leftarrow & \theta - \alpha \nabla_\theta \frac{1}{B} \sum_i \omega(s_i, a_i)(\mathcal{T}_\lambda - Q_{tot}(s_i, a_i; \theta))^2 \\ \phi \leftarrow & \phi - \alpha \nabla_\phi \frac{1}{B} \sum_i (\mathcal{T}_\lambda - \hat{Q}^*(s_i, a_i; \phi))^2 \end{aligned}, \tag{17}$$

where B is the batch size, ω is the weighting function, $\hat{Q}^*$ is the unrestricted joint action value function. As shown in Algorithm 3, the way to compute TD target in WQMIX is different from that in QMIX. Thus, we have

$$\text{FLOPs}_{\text{TD_target}} = B \times (\text{FLOPs}_{\text{Unrestricted-Agent}} + \text{FLOPs}_{\text{Unrestricted-Mix}}). \tag{18}$$

In this paper, we take an experiment on one of two instantiations of QMIX. i.e., OW-QMIX [Rashid et al., 2020a]. Thus, the number of FLOPs in computing loss is

$$\text{FLOPs}_{\text{compute_loss}} = B \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}} + \text{FLOPs}_{\text{Unrestricted-Agent}} + \text{FLOPs}_{\text{Unrestricted-Mix}}). \tag{19}$$

where unrestricted-agent and unrestricted-mix have similar network architectures as Q_{tot} and Q_{tot} to, respevtively. The FLOPs of gradients backward propagation can be given as

$$\text{FLOPs}_{\text{backward_pass}} = B \times 2 \times (\text{FLOPs}_{\text{Agent}} + \text{FLOPs}_{\text{Mix}} + \text{FLOPs}_{\text{Unrestricted-Agent}} + \text{FLOPs}_{\text{Unrestricted-Mix}}). \tag{20}$$

Thus, the FLOPs of training in WQMIX can be computed by

$$\text{FLOPs}_{\text{train}} = B \times (3\text{FLOPs}_{\text{Agent}} + 3\text{FLOPs}_{\text{Mix}} + 4\text{FLOPs}_{\text{Unrestricted-Agent}} + 4\text{FLOPs}_{\text{Unrestricted-Mix}}). \tag{21}$$

B.4.5 Training FLOPs Calculation in RES

Calculations of FLOPs for RES are similar to those in QMIX. The way to update the network parameter in RES is:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_i (\mathcal{T}_\lambda - Q_{tot}(s_i, a_i; \theta))^2, \tag{22}$$

where B is the batch size. Meanwhile, note that the way to compute TD target in RES [Pan et al., 2021] includes computing the *approximate Softmax operator*, we have:

$$\text{FLOPs}_{\text{TD_target}} = B \times (\text{FLOPs}_{\text{Agent}} + n \times m \times (2\text{FLOPs}_{\text{Mix}})), \tag{23}$$

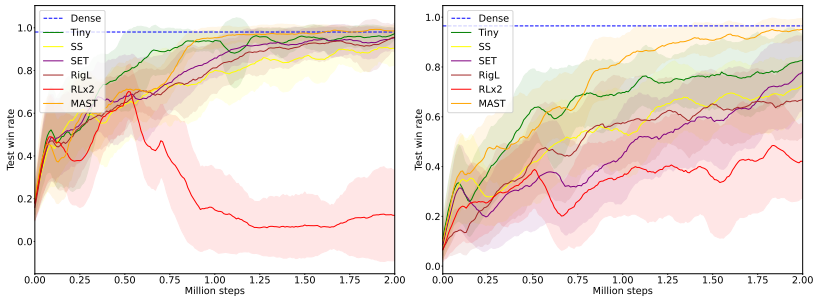
where n is the number of agents, m is the maximum number of actions an agent can take in a scenario. Other terms for updating networks are the same as QMIX. Thus, the FLOPs of training in RES can be computed by

$$\text{FLOPs}_{\text{train}} = B \times (4\text{FLOPs}_{\text{Agent}} + (3 + 2 \times n \times m)\text{FLOPs}_{\text{Mix}}). \tag{24}$$

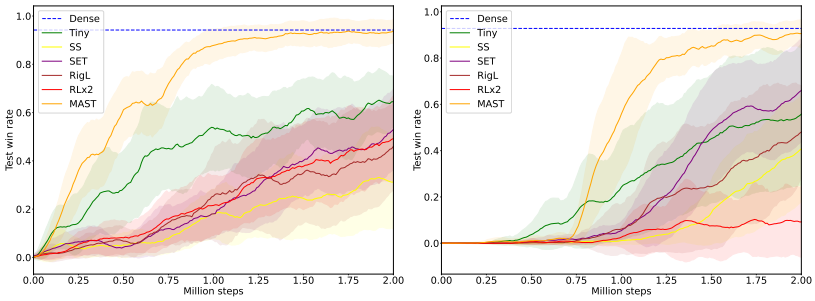
B.5 Training Curves of Comparative Evaluation in Section 4.1

Figure 13, Figure 14, and Figure 15 show the training curves of different algorithms in four SMAC environments. The performance is calculated as the average win rate per episode over the last 20 evaluations of the training. MAST outperforms baseline algorithms on all four environments in all three algorithms. We smooth the training curve by a 1-D filter by `scipy.signal.savgol_filter` in Python with `window_length=21` and `polyorder=2`.

These figures unequivocally illustrate MAST's substantial performance superiority over all baseline methods in all four environments across the three algorithms. Notably, static sparse (`SS`) consistently exhibit the lowest performance on average, highlighting the difficulty of finding optimal sparse network topologies in the context of sparse MARL models. Dynamic sparse training methods, namely `SET` and `RigL`, slightly outperform (`SS`), although their performance remains unsatisfactory. Sparse networks also, on average, underperform tiny dense networks. However, MAST significantly outpaces all other baselines, indicating the successful realization of accurate value estimation through our MAST method, which effectively guides gradient-based topology evolution. Notably, the single-agent method `RLx2` consistently delivers subpar results in all experiments, potentially due to the sensitivity of the step length in the multi-step targets, and the failure of dynamic buffer for episode-form training samples.

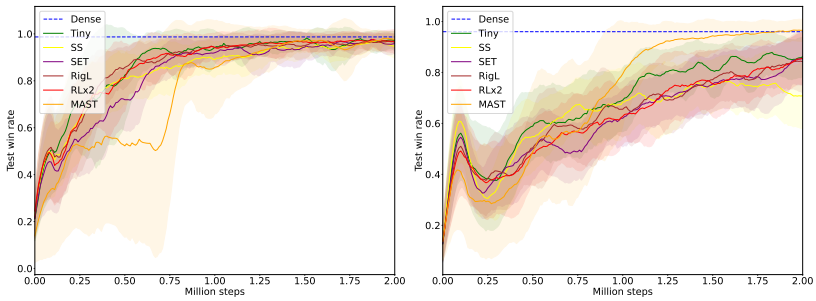


(a) MAST-QMIX 95% on `3m` (b) MAST-QMIX 95% on `2s3z`

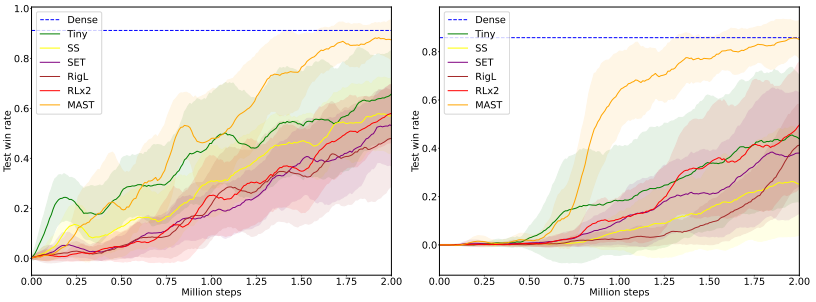


(c) MAST-QMIX 90% on `3s5z` (d) MAST-QMIX 90% on `64zg`

Figure 13: Training processes of MAST-QMIX on four SAMC benchmarks.

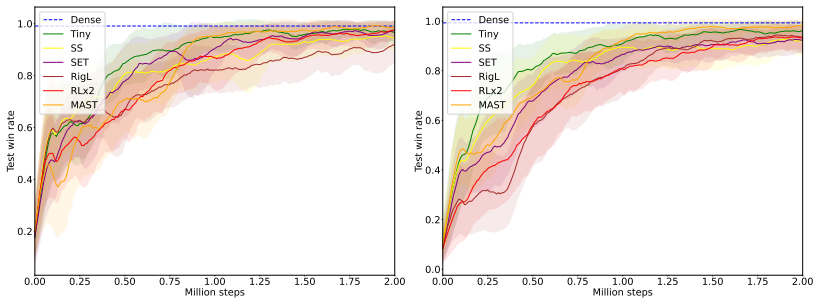


(a) MAST-WQMIX 90% on 3m (b) MAST-WQMIX 90% on 2s3z

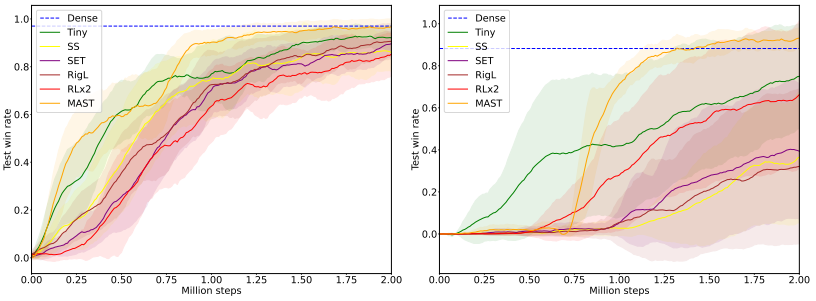


(c) MAST-WQMIX 90% on 3s5z (d) MAST-WQMIX 90% on 64zg

Figure 14: Training processes of MAST-WQMIX on four SAMC benchmarks.



(a) MAST-RES 95% on 3m (b) MAST-RES 90% on 2s3z



(c) MAST-RES 85% on 3s5z (d) MAST-RES 85% on 64zg

Figure 15: Training processes of MAST-RES on four SAMC benchmarks.

B.6 Standard Deviations of Results in Table 1

Table 5 showcases algorithm performance across four SMAC environments along with their corresponding standard deviations. It's important to note that the data in Table 5 is not normalized concerning the dense model. Notably, MAST's utilization of topology evolution doesn't yield increased variance in results, demonstrating consistent performance across multiple random seeds.

Table 5: Results in Table 1 with standard deviations

Alg.	Env.	Tiny(%)	SS(%)	SET(%)	RigL(%)	RLx2(%)	Ours(%)
Q-MIX	3m	96.3±4.3	89.8±7.9	94.1±5.9	93.4±9.5	11.9±20.5	**98.9**±2.0
	2s3z	80.8±12.9	70.4±13.1	74.9±16.4	67.0±14.0	44.2±17.0	**94.6**±4.6
	3s5z	64.2±11.8	32.0±20.3	49.3±16.6	42.6±19.2	47.2±16.2	**93.3**±5.1
	64*	54.0±29.9	37.3±23.4	62.3±21.9	45.2±23.4	9.2±15.0	**90.6**±7.8
	Avg.	73.8±14.7	57.4±16.2	70.1±15.2	62.0±16.5	28.1±17.2	**94.3**±4.9
WQ-MIX	3m	97.0±4.0	95.6±4.0	96.5±3.6	96.5±3.6	96.7±4.3	**97.3**±4.0
	2s3z	86.0±7.9	72.4±12.4	82.5±10.9	83.3±10.3	83.8±9.9	**96.2**±4.2
	3s5z	64.5±17.9	57.0±14.5	51.1±15.0	46.0±20.5	55.4±11.3	**87.6**±6.9
	64*	43.8±27.4	25.4±22.0	37.8±26.2	35.2±16.7	45.3±24.7	**84.4**±8.4
	Avg.	68.5±13.5	62.2±13.0	64.0±13.5	65.8±11.4	70.3±12.5	**91.4**±5.9
RES	3m	96.9±4.1	94.7±4.8	96.4±4.3	90.3±7.4	97.0±3.8	**102.2**±3.2
	2s3z	95.8±3.8	92.2±5.9	92.2±5.5	94.0±5.7	93.4±5.5	**97.7**±2.6
	3s5z	92.2±4.8	86.3±8.8	87.6±5.9	90.0±7.3	83.6±9.2	**96.4**±3.4
	64*	73.5±25.8	34.5±29.6	38.9±32.3	31.1±36.2	64.1±33.8	**92.5**±4.9
	Avg.	89.6±9.6	76.9±12.3	78.8±12.0	76.3±14.1	84.5±13.1	**97.2**±3.5
Avg.		78.7±12.8	65.6±13.9	72.0±13.7	67.8±14.5	61.0±14.3	**94.2**±4.6

B.7 Ablation Study

We conduct a comprehensive ablation study on three critical elements of MAST: hybrid TD(λ) targets, the Soft Mellowmax operator, and dual buffers, specifically evaluating their effects on QMIX and WQMIX. Notably, since MAST-QMIX shares similarities with MAST-RES, our experiments focus on QMIX and WQMIX within the `3s5z` task. This meticulous analysis seeks to elucidate the influence of each component on MAST and their robustness in the face of hyperparameter variations. The reported results are expressed as percentages and are normalized to dense models.

Hybrid TD(λ) We commence our analysis by evaluating different burn-in time T_0 in hybrid TD(λ) in Table 6. Additionally, we explore the impact of different λ values within hybrid TD(λ) in Table 7. These results reveal hybrid TD(λ) targets achieve optimal performance with a burn-in time of $T_0 = 0.75$M and $\lambda = 0.6$. It is noteworthy that hybrid TD(λ) targets lead to significant performance improvements in WQMIX, while their impact on QMIX is relatively modest.

Table 6: Ablation study on burn-in time T_0 in hybrid TD(λ).

Alg.	0	0.45M	0.75M	1M	1.5M	2M
QMIX / RES	93.6	94.3	**97.9**	92.0	92.5	91.5
WQMIX	83.5	88.4	98.0	**98.7**	76.9	70.3
Avg.	88.5	91.3	**97.9**	95.4	84.7	80.9

Table 7: Ablation study on λ in hybrid TD(λ).

Alg.	0	0.2	0.4	0.6	0.8	1
QMIX / RES	91.5	94.7	96.8	96.8	**97.9**	89.4
WQMIX	83.5	83.5	74.7	**98.0**	96.1	87.9
Avg.	87.5	89.1	85.7	**97.4**	97.0	88.6

Soft Mellowmax Operator The Soft Mellowmax operator in Eq.(3) introduces two hyperparameters, α and ω. A comprehensive examination of various parameter configurations is presented

in Table 8, showing that the performance of MAST exhibits robustness to changes in the two hyperparameters associated with the Soft Mellowmax operator.

Additionally, it is worth noting that the Softmax operator is also employed in [Pan et al., 2021] to mitigate overestimation in multi-agent Q learning. To examine the effectiveness of various operators, including max, Softmax, Mellowmax, and Soft Mellowmax, we conduct a comparative analysis in Figure 16. Our findings indicate that the Soft Mellowmax operator surpasses all other baselines in alleviating overestimation. Although the Softmax operator demonstrates similar performance to the Soft Mellowmax operator, it should be noted that the Softmax operator entails higher computational costs.

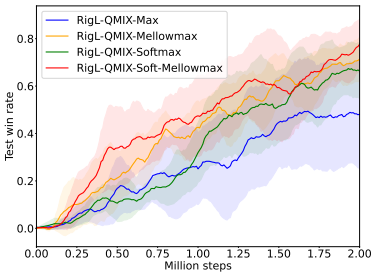


Figure 16: Comparison of different operators.

Table 8: Ablation study on the Soft Mellowmax operator.

Alg.	$\alpha = 1$ $\omega = 10$	$\alpha = 5$ $\omega = 5$	$\alpha = 5$ $\omega = 10$	$\alpha = 10$ $\omega = 5$	$\alpha = 10$ $\omega = 10$
QMIX / RES	97.9	**100.0**	98.9	96.8	97.9
WQMIX	**98.0**	92.3	87.9	92.3	85.7
Avg.	**97.9**	96.1	93.4	94.5	91.8

Dual Buffers In each training step, we concurrently sample two batches from the two buffers, $\mathcal{B}_1$ and $\mathcal{B}_2$. We maintain a fixed total batch size of 32 while varying the sample partitions $b_1 : b_2$ within MAST. The results, detailed in Table 9, reveal that employing two buffers with a partition ratio of $6 : 2$ yields the best performance. Additionally, we observed a significant degradation in MAST's performance when using data solely from a single buffer, whether it be the online or offline buffer. This underscores the vital role of dual buffers in sparse MARL.

Table 9: Ablation study on dual buffers.

Alg.	$8:0$	$6:2$	$5:3$	$3:5$	$2:6$	$0:8$
QMIX / RES	93.6	**99.4**	97.9	97.8	89.6	85.1
WQMIX	64.8	**101.2**	98.0	86.8	95.1	70.3
Avg.	79.2	**100.3**	97.9	92.3	92.4	77.7

B.8 Sensitivity Analysis for Hyperparameters

Table 10 shows the performance with different mask update intervals (denoted as Δ_m) in different environments, which reveals several key observations:

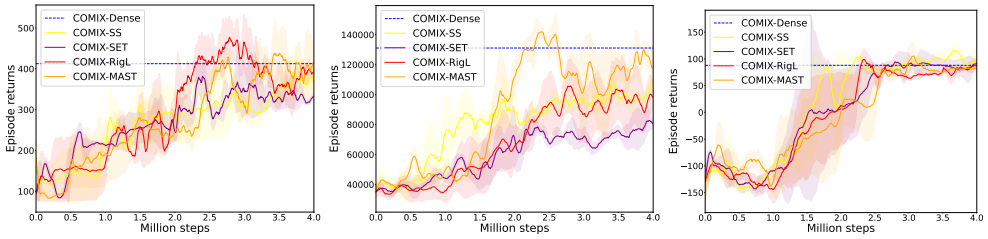
- Findings indicate that a small Δ_m negatively impacts performance, as frequent mask adjustments may prematurely drop critical connections before their weights are adequately updated by the optimizer.
- Overall, A moderate $\Delta_m = 200$ episodes performs well in different algorithms.

Table 10: Sensitivity analysis on mask update interval.

Alg.	$\Delta_m = 20$ episodes	$\Delta_m = 100$ episodes	$\Delta_m = 200$ episodes	$\Delta_m = 1000$ episodes	$\Delta_m = 2000$ episodes
QMIX/RES	99.4%	97.7%	99.0%	**100.6%**	**100.6%**
WQMIX	83.2%	91.9%	**96.1%**	68.1%	71.5%
Avg.	91.3%	94.8%	**97.5%**	84.3%	86.0%

B.9 Experiments on QMIX in Multi-Agent MuJoCo

We compare MAST-QMIX with other baselines on the MAMuJoCo benchmark, and the results are presented in Figure 17. From the figure, we observe that MAST consistently achieves dense performance and outperforms other methods in three environments, except for ManyAgent Swimmer, where the static sparse network performs similarly to MAST. This similarity may be attributed to the simplicity of the ManyAgent Swimmer environment, which allows a random static sparse network to achieve good performance. This comparison demonstrates the applicability of MAST across different environments.

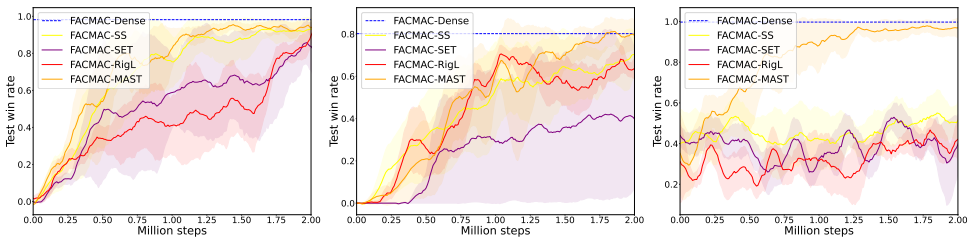


(a) 2-Agent Humanoid (b) 2-Agent HumanoidStandup (c) ManyAgent Swimmer

Figure 17: Mean episode return on different MAMuJoCo tasks with MAST-QMIX.

B.10 Experiments on FACMAC in SMAC

In addition to pure value-based deep MARL algorithms, we also evaluate MAST with a hybrid value-based and policy-based algorithm, FACMAC [Peng et al., 2021], in SMAC. The results are presented in Figure 18. From the figure, we observe that MAST consistently achieves dense performance and outperforms other methods in three environments, demonstrating its applicability across different algorithms.

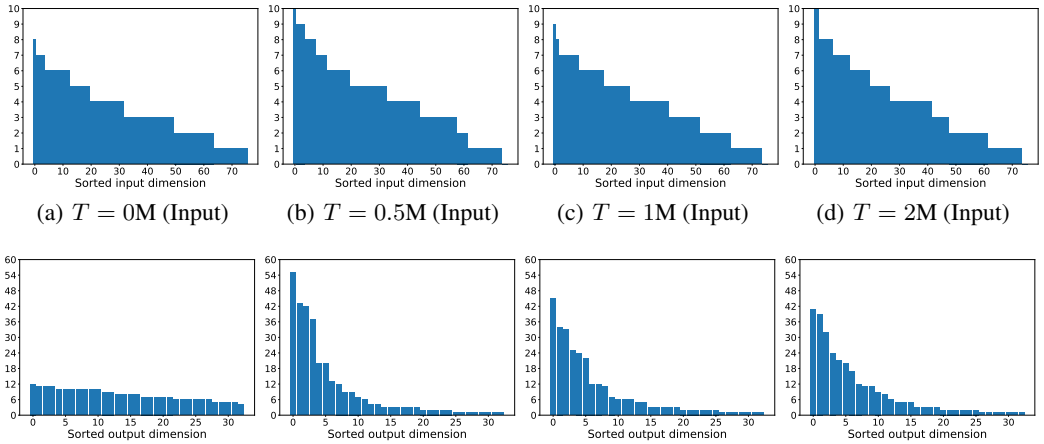


(a) MMM (b) 2c_vs_64zg (c) bane_vs_bane

Figure 18: Mean episode win rates on different SMAC tasks with MAST-FACMAC.

B.11 Visualization of Sparse Masks

We present a series of visualizations capturing the evolution of masks within network layers during the MAST-QMIX training in the `3s5z` scenario. These figures, specifically Figure 20 (a detailed view of Figure 11, Figure 21, and Figure 22, offer intriguing insights. Additionally, we provide connection counts in Figure 23, 24 and 25, for input and output dimensions in each sparse mask, highlighting pruned dimensions. To facilitate a clearer perspective on connection distributions, we sort dimensions based on the descending order of nonzero connections, focusing on the distribution rather than specific dimension ordering. The connection counts associated with Figure 11 in the main paper s given in Figure 19.



(a) $T = 0$M (Input) (b) $T = 0.5$M (Input) (c) $T = 1$M (Input) (d) $T = 2$M (Input)

(e) $T = 0$M (Output) (f) $T = 0.5$M (Output) (g) $T = 1$M (Output) (h) $T = 2$M (Output)

Figure 19: Number of nonzero connections for input and output dimensions in descending order of the sparse layer visualized in Figure 11.

During the initial phases of training, a noticeable shift in the mask configuration becomes evident, signifying a dynamic restructuring process. As the training progresses, connections within the hidden layers gradually coalesce into a subset of neurons. This intriguing phenomenon underscores the distinct roles assumed by individual neurons in the representation process, thereby accentuating the significant redundancy prevalent in dense models.

- Figure 20 provides insights into the input layer, revealing that certain output dimensions can be omitted while preserving the necessity of each input dimension.
- Figure 21 showcases analogous observations, reinforcing the idea that only a subset of output neurons is indispensable, even within the hidden layer of the GRU.
- Figure 22 presents distinct findings, shedding light on the potential redundancy of certain input dimensions in learning the hyperparameters within the hypernetwork.

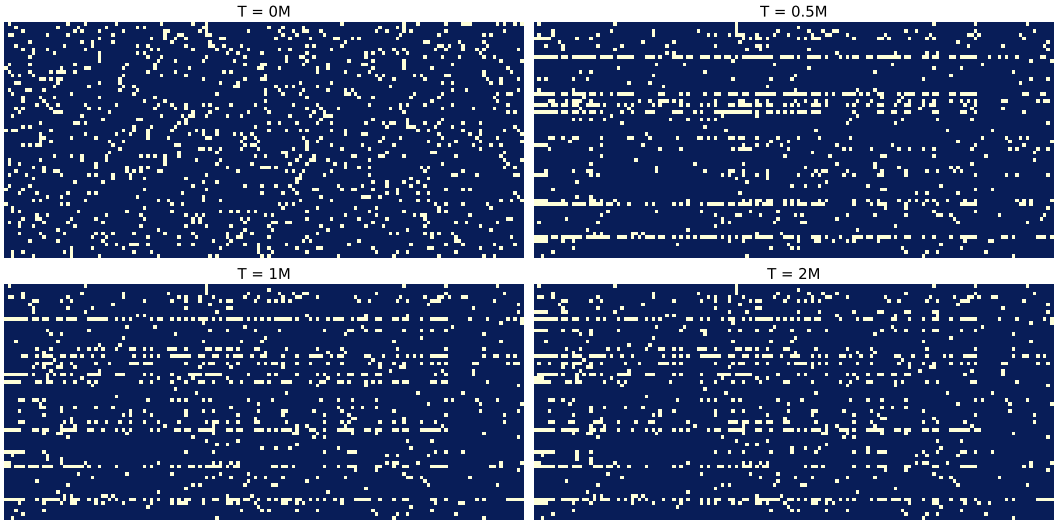


Figure 20: The learned mask of the input layer weight of Q_1. Light pixels in row i and column j indicate the existence of the connection for input dimension j and output dimension i, while the dark pixel represents the empty connection.

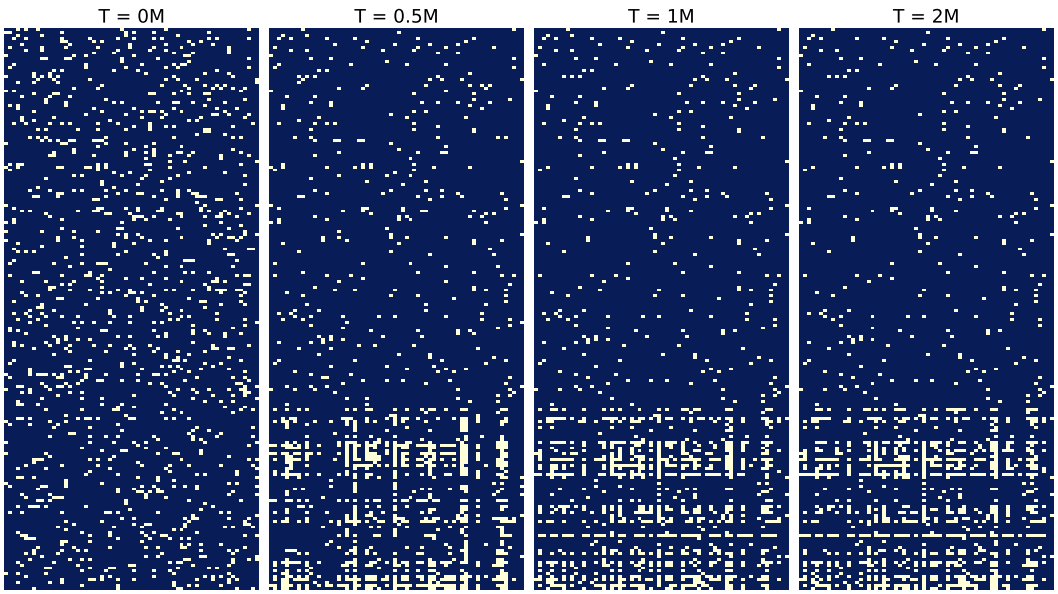


Figure 21: The learned mask of the GRU layer weight of Q_1. Light pixels in row i and column j indicate the existence of the connection for input dimension j and output dimension i, while the dark pixel represents the empty connection.

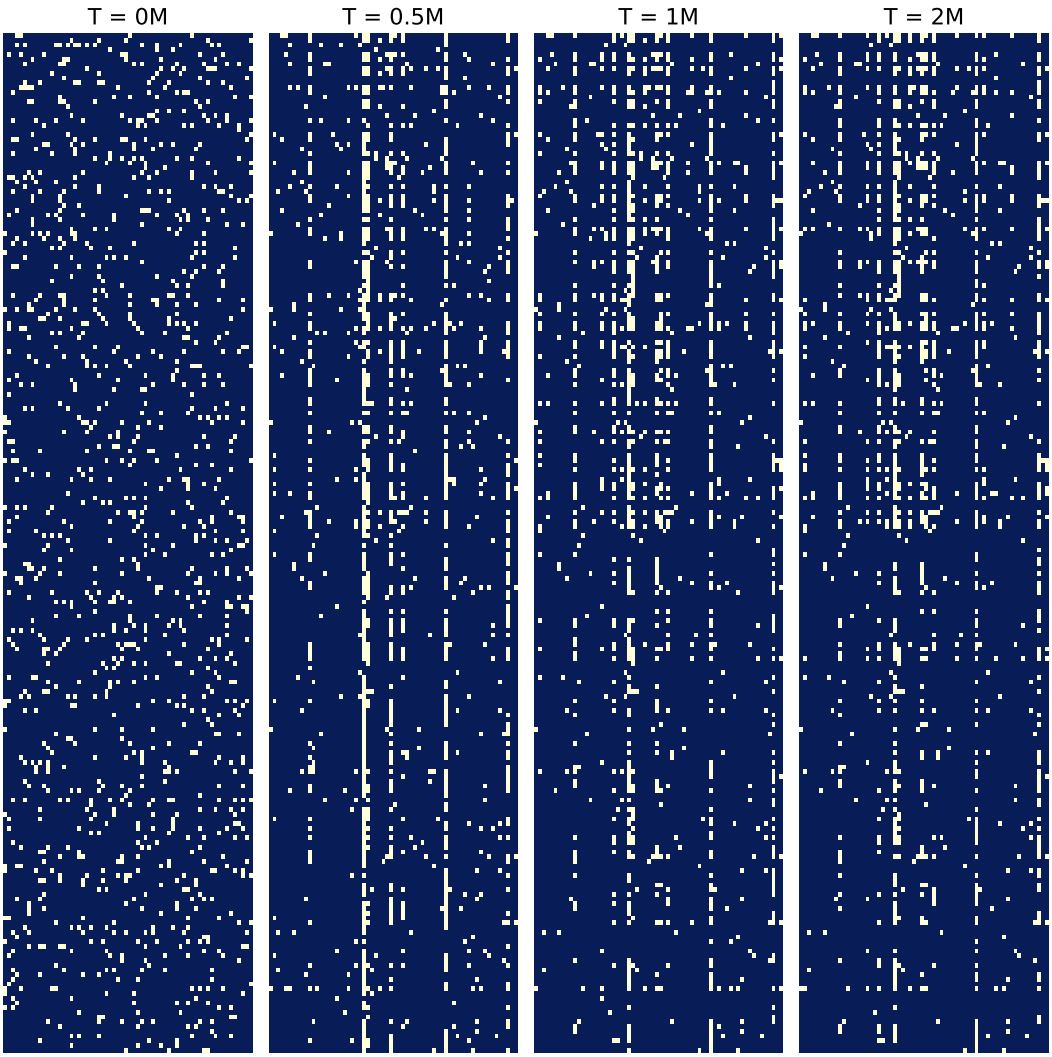
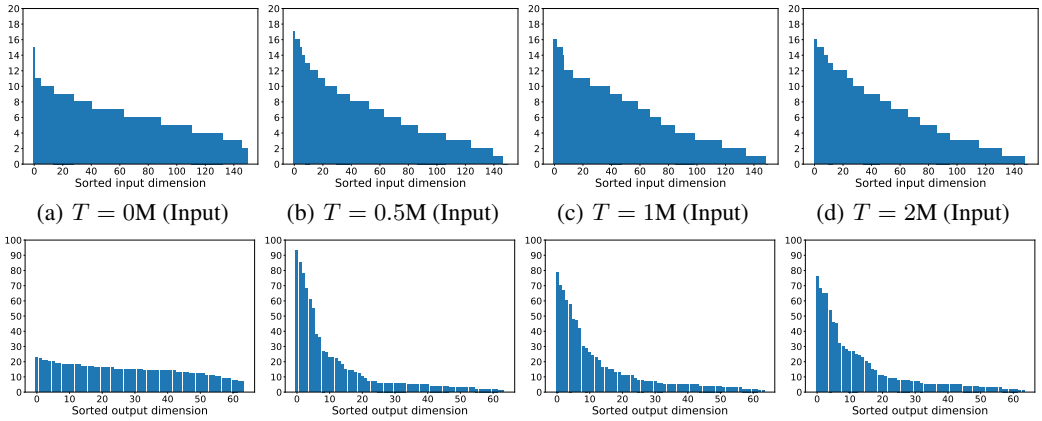


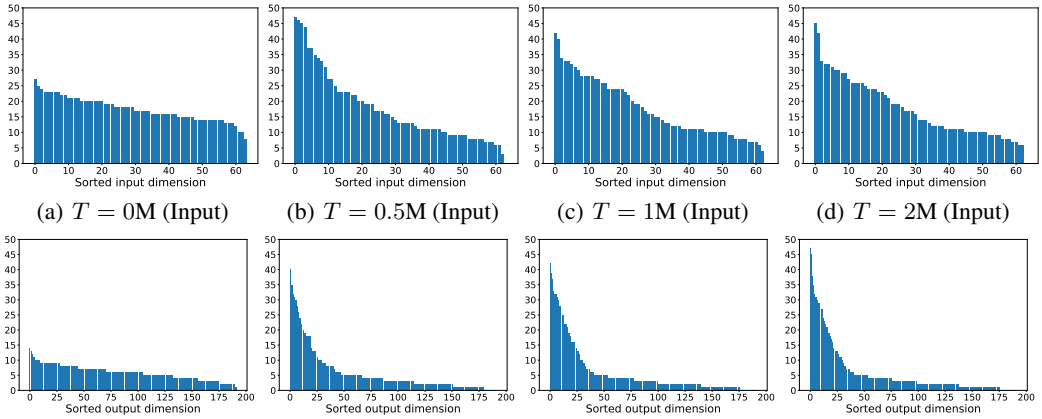
Figure 22: The learned mask of the first layer weight of Hypernetwork. Light pixels in row i and column j indicate the existence of the connection for input dimension j and output dimension i, while the dark pixel represents the empty connection.



(a) $T = 0$M (Input) (b) $T = 0.5$M (Input) (c) $T = 1$M (Input) (d) $T = 2$M (Input)

(e) $T = 0$M (Output) (f) $T = 0.5$M (Output) (g) $T = 1$M (Output) (h) $T = 2$M (Output)

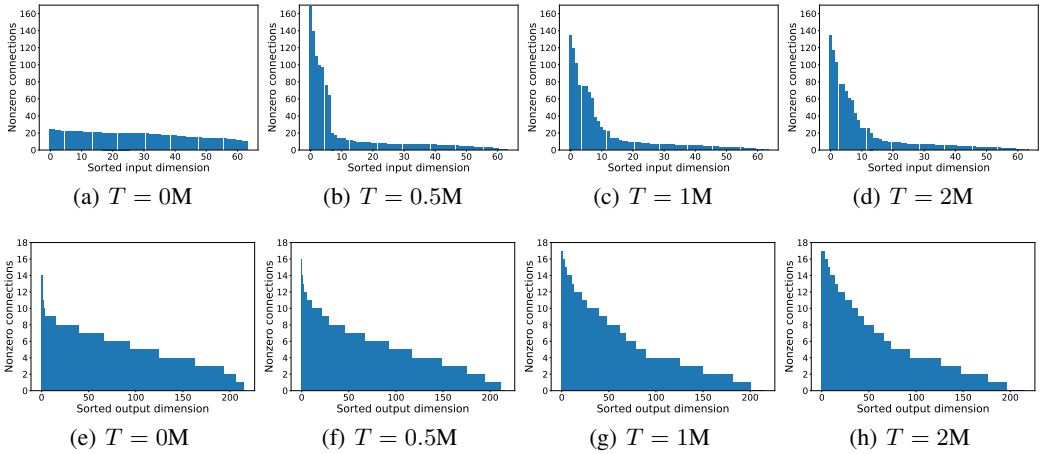
Figure 23: Number of nonzero connections for input and output dimensions in descending order of the sparse layer visualized in Figure 20.



(a) $T = 0$M (Input) (b) $T = 0.5$M (Input) (c) $T = 1$M (Input) (d) $T = 2$M (Input)

(e) $T = 0$M (Output) (f) $T = 0.5$M (Output) (g) $T = 1$M (Output) (h) $T = 2$M (Output)

Figure 24: Number of nonzero connections for input and output dimensions in descending order of the sparse layer visualized in Figure 21.



(a) $T = 0$M (b) $T = 0.5$M (c) $T = 1$M (d) $T = 2$M

(e) $T = 0$M (f) $T = 0.5$M (g) $T = 1$M (h) $T = 2$M

Figure 25: Number of nonzero connections for input and output dimensions in descending order of the sparse layer visualized in Figure 22.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and introduction clearly state the claims that training in multi-agent reinforcement learning involves heavy computational load, and our proposed MAST framework can save up to 20 times FLOPs without performance degradation less than 3% for value-based deep MARL algorithms.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Our paper explores some shortcomings of the MAST framework, such as the multiple hyperparameters. Please refer to Appendix A.6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All our theoretical results are accompanied by clear, complete and correct proofs. All the assumptions we present are clearly stated or referenced.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We ensure that the configurations and details of each experimental result are clearly explained, and all experimental results are reproducible.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code will be open-sourced upon publication of the paper.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We ensure that the parameter configurations and details of each experimental result are clearly explained, such as the network architecture and training hyperparameters. Please refer to B.3.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For each experiment, we presented the results of eight random seeds and calculated their mean and standard deviation. Pleasr refer to B.6

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided the necessary computational resources, hardware setup, and duration required to reproduce the experiments. Please refer to B.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We ensure compliance with all aspects of the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly credited assets used in our paper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have provided a README file alongside our code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.