STAGNANT NEURONS: UNDERSTANDING AND RESTORING PLASTICITY IN MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-Agent Reinforcement Learning (MARL) methods can suffer from loss of plasticity, gradually becoming harder to adapt under sequential task changes. We identify stagnant neurons as a recurring neuron-level pattern associated with impaired adaptation. In value-factorization methods, these units exhibit disproportionately small gradient-based update activity relative to parameter scale and layer-wise peers. Existing plasticity interventions are not designed to identify and repair this pattern directly. We therefore propose **K**nowledge-retentive **N**euron-level Plast**I**city **F**ocusing Inj**E**ction (KNIFE), a diagnosis-conditioned, neuron-local plasticity reallocation method that exactly preserves the network function at the intervention boundary. RUA identifies impaired units, local surgery restores trainable capacity, and decay with pruning recycles temporary capacity. Across SMACv2, SMAC, Predator–Prey, matrix games, and MEAL, KNIFE consistently outperforms the compared plasticity interventions. Beyond value factorization, KNIFE also improves adaptation with MAPPO, MADDPG, and DGN, spanning policy-gradient, actor–critic, and graph-based MARL learners. Additional single-agent PPO results demonstrate that the intervention is not specific to value-factorization training.

1 INTRODUCTION

Multi-Agent Reinforcement Learning (MARL) is widely used in domains such as traffic control Hernandez-Leal et al. (2019) and games Rashid et al. (2018). In practical deployments, environment dynamics or cooperation objectives may change over time, requiring continual adaptation Tomilin et al. (2026). While plasticity loss has been studied extensively in single-agent reinforcement learning Lyle et al. (2023); Nikishin et al. (2023); Dohare et al. (2024), its manifestation and neuron-level mechanisms in MARL remain less understood. This question is particularly important because MARL learners span diverse architectures, including value-factorization, policy-gradient, actor–critic, and graph-based methods Rashid et al. (2018); Wang et al. (2021); Yu et al. (2022); Lowe et al. (2017); Jiang et al. (2020).

We first study this phenomenon in value-factorization methods, where the mixing network provides a controlled setting for analyzing cooperative representation and adaptation. We use the one-step matrix game, Predator–Prey, and a modified SMACv2 benchmark Ellis et al. (2023), in which agents sequentially transfer across newly configured task instances. Adaptation degrades as transfers accumulate: matrix-game bias grows and SMACv2 win rates drop. When returned to a previously mastered task, fine-tuned QMIX Rashid et al. (2018) and QPLEX Wang et al. (2021) fail to recover their earlier performance despite the same training budget, underperforming models trained from scratch on that task.

To characterize this degradation, we analyze neuron-level update dynamics and identify two extreme groups based on relative update activity. *Stagnant neurons* have low relative gradient-based update activity (a low ratio between gradient norm and incoming-weight norm), while *volatile neurons* exhibit the opposite. In our primary value-factorization analysis, stagnant neurons consistently appear across learners and are concentrated in the mixing network. They correlate with slower learning, persist over time, and are distinct from dormant neurons Sokar et al. (2023) and GraMa neurons Liu et al. (2025b), motivating targeted neuron-level intervention.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To mitigate plasticity loss during sequential task transfers, we propose **K**nowledge-retentive **N**euron-level Plast**I**city **F**ocusing Inj**E**ction (KNIFE). KNIFE performs diagnosis-conditioned, neuron-local plasticity reallocation while exactly preserving the network function at the intervention boundary: RUA identifies where plasticity is impaired, local surgery restores trainable capacity, and per-update decay with pruning recycles the temporary capacity. The surgery is realized by knowledge, active, and compensation branches; the knowledge and compensation branches are frozen, while the active branch is re-initialized and trainable. The algorithmic contribution lies in integrating diagnosis, local repair, and recycling rather than in any single branch in isolation.

For value-factorization models, we theoretically show that KNIFE's injection stage satisfies the knowledge invariant++ (KI++) principle proposed in this work; this guarantee is specific to the value-factorization setting and the intervention boundary. Across SMACv2, SMAC, Predator–Prey, matrix games, and MEAL, KNIFE improves adaptation over existing plasticity interventions and reduces the stagnant-neuron ratio. RUA-identified stagnant neurons also provide more effective intervention targets than activation- or gradient-based alternatives. Although our mechanistic analysis focuses on value factorization, KNIFE also improves adaptation when integrated with MAPPO, MADDPG, and DGN, covering policy-gradient, actor–critic, and graph-based MARL learners. Additional single-agent PPO results further demonstrate that the underlying neuron-level intervention is not specific to value-factorization training.

2 Background

2.1 Dec-POMDPs

Cooperative MARL can be modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) Oliehoek & Amato (2016), formally described by the tuple $\langle \mathcal{S}, \{\mathcal{A}_i\}_{i=1}^N, P, r, \{\mathcal{O}_i\}_{i=1}^N, \{\Omega_i\}_{i=1}^N, N \rangle$, where N is the number of agents, $\mathcal{S}$ is a finite set of states, and $\mathcal{A}_i$ is the action set of agent i. At each step t, agent i selects an action $a_i^t \in \mathcal{A}_i$, forming a joint action $\mathbf{a}^t \in \mathcal{A}^N = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$. This joint action leads to a new state $s^{t+1} \sim P(\cdot|s^t, \mathbf{a}^t)$ and a reward r^t. Each agent receives an individual observation $o_i^t \in O_i$ drawn according to $o_i^t \sim \Omega_i(\cdot|s^t)$. The individual policy of agent i is $\pi_i(a_i|\tau_i)$, where $\tau_i = (O_i \times A_i)^*$ is its local action-observation history. The state-action value function for agent i is denoted by $Q_i(\tau_i, a_i)$. The joint action-observation history is denoted as $\tau \in \mathcal{T}^N := \tau_1 \times \ldots \times \tau_N$, on which it conditions the joint policy $\pi =< \pi_1, \ldots, \pi_N >$. The joint policy π has a joint state-action value function $Q_{tot}(\tau, \mathbf{a})$.

2.2 Value Function Factorization

Value factorization is a widely adopted class of MARL methods under centralized training and decentralized execution. An agent network estimates each agent's utility $Q_i(\tau_i, a_i)$ from its local action–observation history, and a mixing network combines these utilities into the joint value $Q_{tot}(\boldsymbol{\tau}, \mathbf{a})$. QMIX Rashid et al. (2018), our primary backbone, conditions its mixing network on the global state and constrains its weights to be non-negative, so that $\partial Q_{tot}/\partial Q_i \geq 0$. This monotonicity allows the joint greedy action to be obtained from decentralized greedy choices. We distinguish the shared agent network from the mixing network because our main interventions are applied only to the latter.

3 Related Work

3.1 Continual Reinforcement Learning (CRL)

Abel et al. Abel et al. (2023) provide a general definition of continual reinforcement learning (CRL) in which the best agents never stop learning. Their framework includes Q-learning in switching MDPs, where an agent repeatedly interacts with environments drawn from a fixed distribution without observing the identity of the active MDP. Our modified SMAC/SMACv2 and matrix-game settings are cooperative multi-agent counterparts of this example, except that we evaluate finite task streams rather than an indefinitely switching process.

Continual MARL setting. We study continuous adaptation of shared policy parameters across a non-stationary sequence of cooperative tasks, without parameter reinitialization. In our modified SMAC/SMACv2 and matrix-game settings, task identity is not provided to the learner. MEAL follows its standard task-head protocol. KNIFE itself never uses task identity or task-boundary signals to trigger intervention. Our objective is efficient adaptation to the active task rather than simultaneous retention of high performance on every previous task; the formal setting is provided in Appendix A.2.

Existing CRL methods generally fall into three categories: regularization-based, architecture-based, and replay-based approaches. Regularization-based methods constrain parameter updates to preserve prior knowledge Kirkpatrick et al. (2017); Zenke et al. (2017); Aljundi et al. (2018); Lewandowski et al. (2025). Architecture-based methods mitigate interference through structural modularization Mallya & Lazebnik (2018); Liu et al. (2025a), while replay-based methods rehearse past experience Atkinson et al. (2021); Wang et al. (2025). Continual MARL further requires both individual policies and cooperative strategies to adapt across tasks Jo et al. (2026). MEAL Tomilin et al. (2026) shows that directly applying continual learning methods to MARL can remain challenging.

3.2 Plasticity Loss

We focus on plasticity injection methods that modify selected network parameters to maintain learning capacity. CBP Dohare et al. (2021) periodically replaces parts of a network, while ReDo Sokar et al. (2023) identifies and resets neurons with low activation scores. ReGraMa Liu et al. (2025b) instead uses gradient-based activity to identify weakly updated neurons. Relatedly, Lyle et al. Lyle et al. (2024) show that parameter-norm growth can interact with scale invariance to reduce the effective learning rate, motivating measurements that relate update magnitude to parameter scale. RUA provides such a neuron-wise, layer-relative statistic without relying on activation dormancy. Other interventions include last-layer resets Nikishin et al. (2022), continual backpropagation Dohare et al. (2024), replay-ratio-aware resets Yang et al. (2024), and ReBorn's perturbation of dormant neurons in MARL mixing networks Qin et al. (2024).

Plasticity Injection (PI) Nikishin et al. (2023) restores plasticity by duplicating the last K layers. PI and KNIFE both introduce fresh and frozen capacity that preserves the function at injection; KNIFE differs in the granularity and allocation of this capacity by combining RUA-based target selection, neuron-local repair, and subsequent recycling through decay and pruning. KNIFE is developed primarily for MARL value factorization methods Hernandez-Leal et al. (2019); Rashid et al. (2018; 2020); Shen et al. (2022; 2023); Yang et al. (2020); Hu et al. (2021); experiments with MADDPG Lowe et al. (2017), MAPPO Yu et al. (2022), and DGN Jiang et al. (2020) demonstrate integration across policy-gradient, actor–critic, and graph-based MARL learners. Detailed comparisons with PI and its initialization strategy are provided in Appendix A.6 and Appendix D.7.

4 The Stagnant Neuron in MARL

In this section, we study neuron-level plasticity loss in MARL using value-factorization methods as our primary analysis setting. Their mixing network provides a controlled coordination bottleneck for examining cooperative representation and adaptation. Across benchmarks and value-factorization learners, we identify stagnant neurons, whose gradient-based update activity is disproportionately small relative to parameter scale and layer-wise peers. These neurons are concentrated in the mixing network, and controlled interventions show that repairing them improves adaptation during task transfers. Extensions to fully connected actor and critic layers in other MARL learners are evaluated in Appendix D.5.

4.1 The Plasticity Loss during Sequential Task Transfers

We study the plasticity loss of MARL during sequential task transfers by modifying a one-step two-agent matrix game and the StarCraft Multi-Agent Challenge v2 (SMACv2) Ellis et al. (2023) for continual learning scenarios.

In the one-step matrix game, each payoff matrix defines a task. After learning a sequence of tasks and returning to the initially mastered Task 1, QMIX retains a high matrix approximation bias rather than recovering its earlier solution (Figure 1(a); lower is better). At the same time, the average gradient

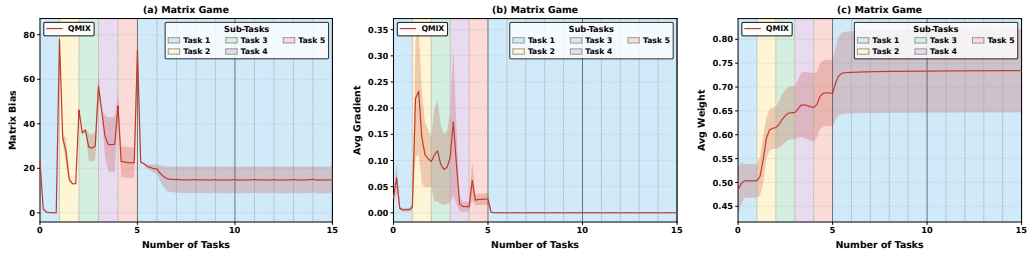

Figure 1: Plasticity loss in the sequential matrix game. After learning Tasks 1–5 and returning to Task 1, QMIX retains a high matrix approximation bias (a), while the average gradient norm approaches zero (b) and the average weight norm remains large (c).

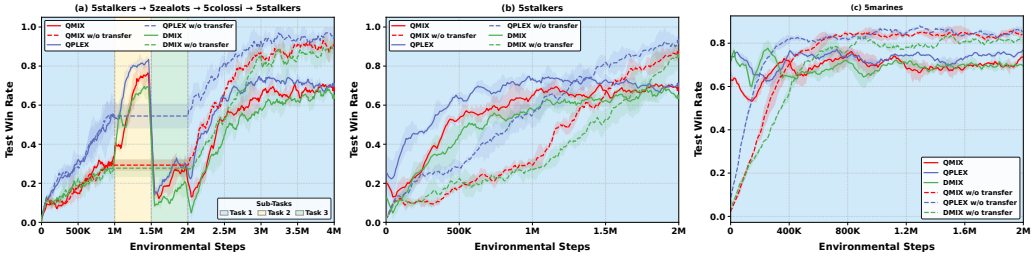

Figure 2: Plasticity loss in modified SMACv2. (a) Protoss training sequence: Task 1 (1M steps), Tasks 2–3 (0.5M each), then Task 1 (2M); dashed segments denote w/o-transfer baselines that skip Tasks 2–3. (b) Revisited Task 1, shifted to 0–2M steps, versus single-task training with the same budget. (c) Corresponding comparison for the initial Terran task.

norm approaches zero while the average weight norm remains large (Figure 1(b–c)), revealing low update activity relative to parameter scale and impaired re-adaptation. Full task matrices and training budgets are provided in Appendix D.1.

We additionally evaluate a cyclic setting that repeatedly revisits multiple previous tasks, providing a more conventional continual-learning protocol; the results show the same progressive degradation in re-adaptation (Appendix D.3.2).

Figure 2 compares re-adaptation to the initial task with single-task training under the same target-task budget. QMIX, QPLEX, and DMIX re-adapt substantially more slowly and reach lower final win rates after sequential transfer, consistently across Protoss and Terran. The task identities and complete sequences are detailed in Appendix Figure A1.

Together, the matrix-game and SMACv2 results reveal a progressive degradation in re-adaptation capacity rather than ordinary forgetting alone: after sequential transfers, a previously trained model can become harder to re-train on a formerly mastered task than a freshly trained counterpart under the same target-task budget. We refer to this degradation of learning ability as *plasticity loss*.

4.2 Stagnant Neurons

To understand potential sources of plasticity loss, we examine neurons in neural networks. We define a few metrics and neurons as follows.

Definition 1 (Relative Update Activity (RUA))**.** *For a neural network layer ℓ, let w_i^ℓ denote the incoming weight vector of neuron i and $g_i^\ell = \nabla_{w_i^\ell}\mathcal{L}$ denote its gradient. We define the **RUA**:*

$$\mathrm{UA}_i^\ell = \frac{\|g_i^\ell\|_2}{\|w_i^\ell\|_2 + \epsilon}, \quad \overline{\mathrm{UA}}_i^\ell = \frac{1}{T}\sum_{t=1}^{T} \mathrm{UA}_i^\ell, \quad \mathrm{RUA}_i^\ell = \frac{\overline{\mathrm{UA}}_i^\ell}{\mathbb{E}_{j\in\ell}[\overline{\mathrm{UA}}_j^\ell]} \tag{1}$$

where ϵ is a small constant for numerical stability. $\overline{\mathrm{UA}}_i^\ell$ is the average gradient-based activity over time interval T.

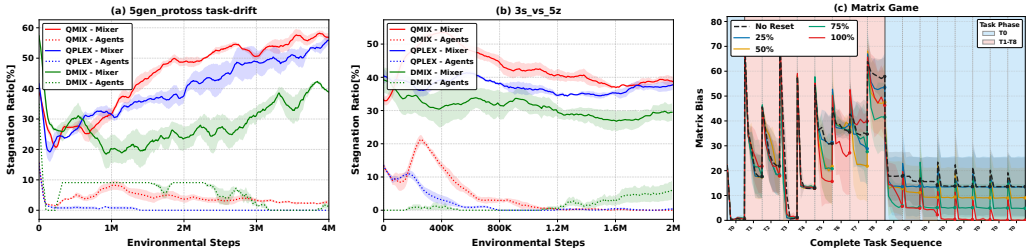

Figure 3: Stagnant-neuron prevalence and repair effects. Ratios in agent and mixing networks for (a) SMACv2 5gen_protoss Task Drift and (b) SMAC 3s_vs_5z, showing greater concentration in the mixing network. (c) Mean recovery bias after repairing varying fractions of naturally detected stagnant neurons in the one-step matrix game.

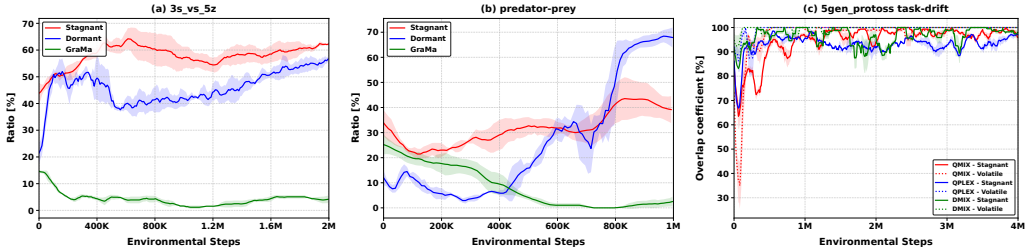

Figure 4: Ratios of stagnant, dormant, and GraMa neurons in (a) SMAC 3s_vs_5z and (b) Predator–prey. (c) The overlap of stagnant and volatile neurons in SMACv2 5gen_protoss task drift.

RUA characterizes neuron-wise relative gradient-based update activity by comparing gradient magnitude against parameter scale and layer-wise peers.

Definition 2 (Stagnant Neuron)**.** *A neuron i in layer ℓ is a stagnant neuron if its score* $\mathrm{RUA}_i^\ell < \alpha$ *(i.e., 0.25).*

Low RUA therefore indicates that a neuron receives disproportionately weak gradient-based updates relative to its current parameter scale and layer-wise peers, a pattern associated with inefficient adaptation. Low RUA does not require a large weight norm: it may arise from a small gradient, a large parameter scale, or their interaction. Appendix Table A7 separates these factors using the GraMa small-gradient criterion and shows that 55.9% of detected stagnant neurons are in the low-weight group. Thus, large weights contribute to stagnation but are not a necessary condition; the defining characteristic is the gradient–parameter imbalance captured by RUA.

Definition 3 (Volatile Neuron)**.** *A neuron i in layer ℓ is a volatile neuron if its score* $\mathrm{RUA}_i^\ell > \beta$.

Volatile neurons lie at the opposite, high-RUA tail and undergo disproportionately large relative updates compared with their layer-wise peers. We treat them as a complementary repair target rather than the main focus of our diagnosis, since jointly repairing stagnant and volatile neurons further reduces stagnation and improves adaptation.

Figure 3 (a) depicts the ratio of stagnant neurons for QMIX, QPLEX, and DMIX in the agent network and the mixing network. We find that stagnant neurons concentrate heavily in the mixing network. Interestingly, these stagnant neurons emerge not only in continual tasks but also in single-task learning scenarios. As shown in Figure 3 (b), stagnant neurons still accumulate in the 3s_vs_5z scenario of SMAC Samvelyan et al. (2019), where agents learn one task only. This pattern is not unique to hypernetworks Ha et al. (2017) used in MARL, but also appears with MLP and CNN mixing architectures. Its accumulation is accompanied by decreases in Weight Spectra Entropy and gradient activity, indicating co-occurring reductions in representational diversity and optimization activity. Detailed structural ablations and entropy analyses are provided in Appendix D.6.

Stagnant neurons are associated with impaired re-adaptation in MARL. To further examine the effect of naturally occurring stagnant neurons, we detect stagnant neurons using RUA and

intervene on $0\%, 25\%, 50\%, 75\%$, and 100% of them in the one-step matrix game while keeping the remaining training procedure unchanged. We repeat each condition with 10 random seeds and report 95% confidence intervals. As shown in Figure 3(c), the mean recovery bias decreases from 18.33 ± 2.86 without repair to 3.94 ± 0.82 with full repair, revealing a dose–response relationship between stagnant-neuron recovery and re-adaptation. Complete results are reported in Appendix Tables A5 and A6.

We additionally provide controlled gradient-manipulation studies in Appendix D.2, where artificially increasing the number of stagnant neurons leads to larger matrix bias. The influence of volatile neurons is further analyzed in Appendix D.2, Figure A3.

Difference from Other Neuron Criteria The stagnant neuron is different from dormant neurons Sokar et al. (2023) and GraMa neurons Liu et al. (2025b), which have been proposed as indicators of plasticity loss in RL. Figure 4 (a) and (b) depict the ratios of different neurons in the mixing network of QMIX for the 3s_vs_5z and the predator-prey environments. Their curves exhibit different trends, indicating differences among these neurons, even in this non-continual setting. Moreover, Figure 6 shows that methods designed around dormant or GraMa neurons do not reduce stagnation as effectively in our evaluated settings.

We study whether the detected stagnant/volatile neurons persist across time. To this end, at each iteration (every time interval), we calculate the overlap coefficient between the current and previous iterations for stagnant/volatile neurons. As shown in Figure 4 (c), there are large overlaps between stagnant neurons across iterations, as well as the volatile neurons. This indicates that once a neuron becomes stagnant/volatile, it will remain the same type of neuron over time with high probability. Their persistence motivates evaluating targeted surgical intervention as a possible way to alleviate plasticity loss.

5 KNIFE: A PLASTICITY INJECTION METHOD

We propose a MARL plasticity injection method dubbed KNIFE, which stands for **K**nowledge-retentive **N**euron-level Plast**I**city **F**ocusing Inj**E**ction. KNIFE implements diagnosis-conditioned, neuron-local plasticity reallocation through three operations: identification, injection, and pruning. Identification uses RUA to detect the set of neurons $\mathcal{S}^{\ell}_{\text{repair}}$ whose plasticity is impaired. At injection, local surgery restores trainable capacity while exactly preserving the pre-intervention network function. Pruning recycles the temporary capacity: after every learner update, the frozen knowledge and compensation parameters are multiplied by γ, and branches whose parameters are close to zero are removed. Identification and injection are performed at a fixed interval of $T = 0.2$M environment steps in all environments. The global schedule starts at the beginning of Task 0 and does not require task-boundary signals. A schematic plot of KNIFE is depicted in Figure 5.

5.1 IDENTIFICATION OPERATION

The identification operation identifies the set of neurons $\mathcal{S}^{\ell}_{\text{repair}}$ that receive neuron surgery in the plasticity injection operation, where $\mathcal{S}^{\ell}_{\text{repair}} = \mathcal{S}^{\ell}_{\text{stagnant}} \cup \mathcal{S}^{\ell}_{\text{volatile}}$, $\mathcal{S}^{\ell}_{\text{stagnant}} = \{i \mid \text{RUA}^{\ell}_i < \alpha\}$, $\mathcal{S}^{\ell}_{\text{volatile}} = \{i \mid \text{RUA}^{\ell}_i > \beta\}$.

5.2 PLASTICITY INJECTION OPERATION

For each neuron i in $\mathcal{S}^{\ell}_{\text{repair}}$, we apply a plasticity injection surgery. This surgery aims to inject plasticity while retaining the learned cooperation knowledge. For neuron i, we replace it with a composite neuron consisting of an active neuron, a knowledge neuron, and a compensation neuron. They all take the same input. Their outputs are summed and used as input to the other neurons. The weights of the knowledge and the compensator neurons are frozen, whereas the weights of the active neurons are not. The knowledge neuron is used to preserve previously learned knowledge, and the compensation neuron is used to compensate for the output of the active neuron.

Only selected neurons are expanded temporarily; frozen branches are subsequently pruned and reused (Appendix D.10).

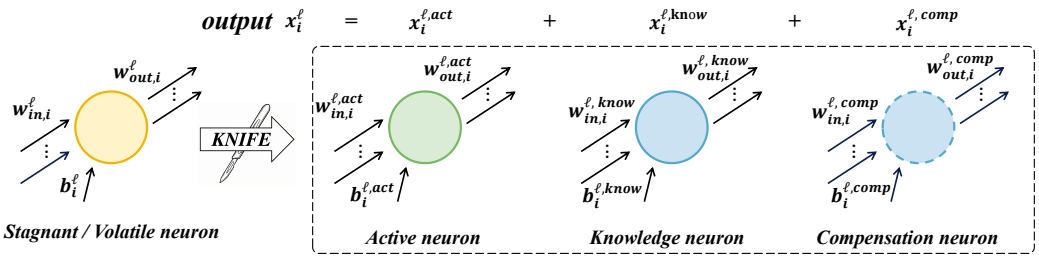

Figure 5: KNIFE identifies plasticity-impaired neurons, performs neuron-local surgery that preserves the function at injection, and recycles the temporary frozen branches through decay and pruning.

Let a_i^ℓ denote the activation (a scalar) of neuron i in layer ℓ as $a_i^\ell = \sigma\big((\mathbf{w}_{in,i}^\ell)^\top \mathbf{x}^{\ell-1} + b_i^\ell\big)$, where $\mathbf{x}^{\ell-1}$ is the input vector to layer ℓ, $\mathbf{w}_{in,i}^\ell$ and b_i^ℓ are the incoming weights and bias, σ is the activation function. The output $\mathbf{x}_i^\ell$ of neuron i to the next layer is defined $\mathbf{x}_i^\ell = a_i^\ell \mathbf{w}_{out,i}^\ell$, where $\mathbf{w}_{out,i}^\ell$ is the output weight of neuron i.

As the learned knowledge is encoded in neural networks, to preserve the knowledge, it is important to ensure that $\mathbf{x}_i^\ell$ remains the same after the plasticity injection surgery. The surgery splits the original neuron's output $\mathbf{x}_i^\ell$ into three parallel branches:

$$\mathbf{x}_i^\ell = \mathbf{x}_i^{\ell,\text{know}} + \mathbf{x}_i^{\ell,\text{active}} + \mathbf{x}_i^{\ell,\text{comp}}$$

$\mathbf{x}_i^{\ell,\text{know}}$, $\mathbf{x}_i^{\ell,\text{active}}$, and $\mathbf{x}_i^{\ell,\text{comp}}$ are the output of the knowledge, the active, and the compensation neuron. We describe the details of these outputs and neurons in the following paragraphs.

Knowledge neuron (frozen). KNIFE copies the original stagnant neuron and freezes its parameters to rigorously preserve previously acquired cooperative capabilities: $\mathbf{w}_{in,i}^{\ell,\text{know}} \leftarrow \mathbf{w}_{in,i}^\ell$, $b_i^{\ell,\text{know}} \leftarrow b_i^\ell$, $\mathbf{w}_{out,i}^{\ell,\text{know}} \leftarrow \mathbf{w}_{out,i}^\ell$.

Active neuron (state-aware initialization). The active branch is freshly initialized, with a positive input bias and an output scale matched to the subsequent layer to encourage early activation and gradient flow. Exact initialization formulas and the corresponding ablation are provided in Appendix D.7.

Compensation neuron (frozen). We construct a compensation branch to compensate the active branch at the injection step by sharing the same incoming parameters and negating the outgoing weights $\mathbf{w}_{in,i}^{\ell,\text{comp}} \leftarrow \mathbf{w}_{in,i}^{\ell,\text{active}}$, $b_i^{\ell,\text{comp}} \leftarrow b_i^{\ell,\text{active}}$, $\mathbf{w}_{out,i}^{\ell,\text{comp}} \leftarrow -\mathbf{w}_{out,i}^{\ell,\text{active}}$

Knowledge Retention of Neuron Surgery After the neuron surgery, we update only the *active* neurons by gradient descent, and freeze the two other neurons by masking their gradients to zero. The output of the new big neuron is calculated as $\mathbf{x}_i^{\ell,\text{know}} + \mathbf{x}_i^{\ell,\text{active}} + \mathbf{x}_i^{\ell,\text{comp}} = \mathbf{x}_i^{\ell,\text{know}} = \mathbf{x}_i^\ell$. Thus, KNIFE preserves the network function at the injection boundary. The formal preservation definitions, architectural scope, theorem, and proof are provided in Appendix A.3–A.4 and Theorem 1.

This construction separates function preservation at injection from subsequent adaptation. As the frozen branches decay, the active branch progressively assumes their role, enabling a gradual handover rather than an abrupt reset.

5.3 Pruning Operation

After each learner update, frozen knowledge and compensation parameters are multiplied by γ and pruned once their norms fall below the threshold, releasing capacity. Thus, T controls detection and injection, whereas γ controls per-update decay. Composite branches are excluded from stagnant-neuron detection; after pruning, the active branch becomes a regular trainable neuron and can be repaired again if needed. This cycle limits persistent model growth while allowing plastic capacity to be refreshed repeatedly.

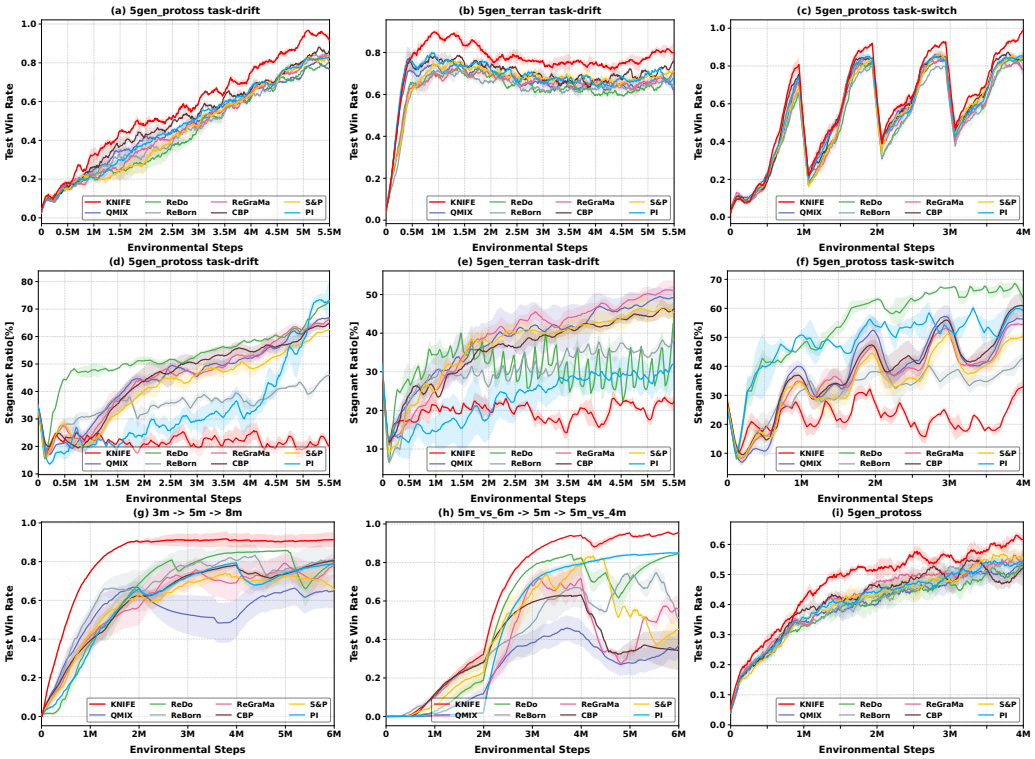

Figure 6: KNIFE performance and stagnant-neuron ratios across task-drift, task-switch, scaling-up/down, and standard single-task settings.

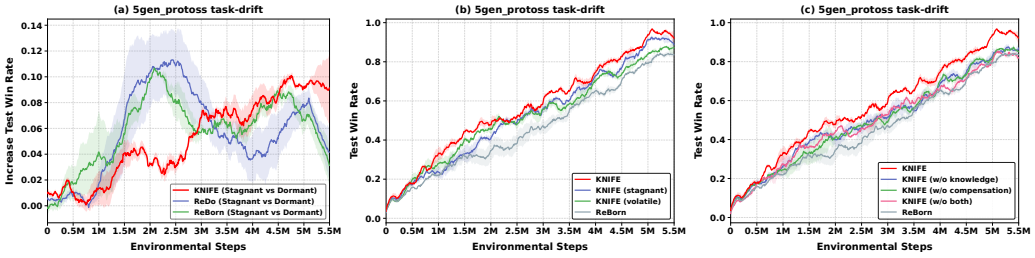

Figure 7: Ablations on intervention targets, stagnant/volatile repair, and knowledge-retentive surgery.

6 Empirical Evaluations

6.1 Experimental Setup

Setup. We evaluate KNIFE on SMAC/SMACv2, Predator–Prey, the one-step matrix game, and MEAL against ReDo, ReBorn, ReGraMa, S&P, CBP, ReSet, and PI. All methods share the same backbone, training budget, and seed protocol; periodic methods use a common 0.2M-step interval. All reported performance results use at least five independent random seeds and report 95% confidence intervals. Unless otherwise specified, results average five seeds; experiments using more seeds are explicitly identified. Full settings and tuning policies are provided in Appendix D.1. Additional 20- and 30-seed robustness evaluations are reported in Appendix D.13.

6.2 Main Results on Continual MARL Benchmarks

Across Task Drift and Task Switch, KNIFE achieves higher win rates and lower stagnant-neuron ratios than the compared plasticity interventions (Figure 6(a–f)). It also remains effective under

Table 1: Average normalized performance on MEAL Level-1.

Setting	QMIX	ReDo	ReBorn	ReGraMa	S&P	CBP	PI	KNIFE
Easy	1.64 ± 0.08	1.63 ± 0.09	1.59 ± 0.08	1.71 ± 0.08	1.63 ± 0.09	1.70 ± 0.08	1.71 ± 0.19	$\mathbf{1.85 \pm 0.08}$
Medium	1.39 ± 0.10	1.49 ± 0.04	1.45 ± 0.12	1.50 ± 0.13	1.43 ± 0.05	1.44 ± 0.12	1.52 ± 0.01	$\mathbf{1.70 \pm 0.09}$

Table 2: KNIFE across MARL learner families on 5gen_protoss task drift (win rate %).

Method	Baseline	+ReDo	+ReBorn	+KNIFE
MAPPO	18.84 ± 2.59	20.26 ± 4.25	17.61 ± 0.91	$\mathbf{25.44 \pm 2.69}$
MADDPG	20.13 ± 2.66	10.48 ± 5.26	29.16 ± 2.27	$\mathbf{34.55 \pm 3.10}$
DGN	17.48 ± 1.66	20.58 ± 2.32	19.85 ± 2.10	$\mathbf{22.73 \pm 1.04}$

scaling-up/down and standard single-task learning (Figure 6(g–i)). The gain is not explained by additional capacity alone, as wider QMIX and ReBorn variants remain below KNIFE (Appendix D.9).

Table 1 summarizes mean performance with 95% confidence intervals over five seeds; task-wise results are reported in Appendix Tables A10 and A11.

On MEAL Level-1, KNIFE achieves the highest average normalized return on both Easy and Medium (Table 1). The advantage is already clear on Easy and becomes more pronounced on Medium, where KNIFE improves over the strongest competing baseline by 11.8% and over QMIX by 22.3%. Under more severe plasticity failure, KNIFE recovers to near-zero bias in the sequential matrix game while the baselines fail to re-adapt (Appendix Figure A8).

6.3 Generalization across MARL Learners

Our mechanistic analysis uses value factorization, while Table 2 demonstrates that KNIFE is not tied to a mixer: it consistently improves MAPPO, MADDPG, and DGN, with additional QPLEX and DMIX results in Appendix Figure A14. KNIFE also improves single-agent PPO from 1.19 to 1.37 on MEAL, demonstrating applicability beyond MARL in this evaluation (Appendix D.4.3). The gain is largest for MADDPG ($+14.42$ percentage points over its baseline), while MAPPO and DGN improve by 6.60 and 5.25 points, respectively, indicating that the benefit is not confined to a single optimization paradigm.

6.4 Neuron Analysis and Ablations

Controlled comparisons separate target selection from repair (Figure 7(a) and Appendix Table A18): with the same KNIFE surgery, RUA-selected stagnant neurons outperform alternative targets, while with identical RUA targets, KNIFE outperforms ReDo and ReBorn. Figure 7(b–c) further shows complementary benefits from volatile-neuron repair and the knowledge/compensation branches that preserve the function at injection. Taken together, these ablations indicate that KNIFE's gains depend on both where and how plasticity is restored: RUA selects effective targets, while the preserved branches retain the current function as the active branch recovers trainable capacity. These two controlled axes rule out a single-factor explanation: fixed-target results isolate the repair benefit, whereas fixed-surgery results isolate the value of RUA-based target selection.

The scope and limitations of KNIFE are discussed in Appendix E.3.

7 Conclusion

We identify stagnant neurons as actionable targets for plasticity restoration in MARL and propose KNIFE, a neuron-local intervention that exactly preserves the network function at injection and improves adaptation across benchmarks and learner families. Across task drift, task switching, structural scaling, and standard learning, KNIFE improves adaptation while reducing stagnation. Controlled ablations support both RUA-based targeting and injection-time function preservation, while gains across multiple learner families demonstrate applicability beyond value factorization.

AI USE DISCLOSURE

Generative AI tools were used for language polishing, literature retrieval and discovery, and assistance with executing computational experiments. All experimental configurations, methodological decisions, analyses, and scientific conclusions were determined and verified by the authors.

REPRODUCIBILITY STATEMENT

All reported performance results use at least five independent random seeds and 95% confidence intervals. Appendix D.1 specifies the task definitions, training budgets, architectures, baseline configurations, intervention schedules, and hyperparameters, while Appendix D.13 provides higher-seed robustness checks. The provided algorithm and experimental details are intended to support independent reproduction of the reported results.

ETHICS STATEMENT

This work uses simulated benchmarks and does not involve human participants or personal data. Improved adaptation in MARL may support more robust coordination, but it could also be misused in unsafe or insufficiently monitored autonomous systems. Our results do not establish readiness for real-world deployment; such use requires domain-specific safety validation, continuous monitoring, and appropriate human oversight.

REFERENCES

David Abel, André Barreto, Benjamin Van Roy, Doina Precup, Hado P van Hasselt, and Satinder Singh. A definition of continual reinforcement learning. *Advances in Neural Information Processing Systems*, 36:50377–50407, 2023.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 139–154, 2018.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *NeurIPS*, 33:3884–3894, 2020.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. *Neurocomputing*, 428:291–307, 2021.

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. In *NeurIPS*, 2023.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *ICLR*, 2017.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019. doi: 10.1007/s10458-019-09421-1.

Siyi Hu, Fengda Zhu, Xiaojun Chang, and Xiaodan Liang. Updet: Universal multi-agent reinforcement learning via policy decoupling with transformers. In *ICLR*, 2021.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *ICLR*, 2020.

Yonghyeon Jo, Sunwoo Lee, and Seungyul Han. Retaining suboptimal actions to follow shifting optima in multi-agent reinforcement learning. In *ICLR*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Alex Lewandowski, Michał Bortkiewicz, Saurabh Kumar, András György, Dale Schuurmans, Mateusz Ostaszewski, and Marlos C Machado. Learning continually by spectral regularization. In *International Conference on Learning Representations*, 2025.

Jiashun Liu, Johan Obando-Ceron, Aaron Courville, and Ling Pan. Neuroplastic expansion in deep reinforcement learning. In *International Conference on Learning Representations*, 2025a.

Jiashun Liu, Zihao Wu, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Ling Pan. Measure gradients, not activations! enhancing neuronal activity in deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-2659.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*, pp. 6379–6390, 2017.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *ICML*, pp. 23190–23211, 2023.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:106440–106473, 2024.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron C. Courville. The primacy bias in deep reinforcement learning. In *ICML*, pp. 16828–16847, 2022. URL `https://proceedings.mlr.press/v162/nikishin22a.html`.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. In *NeurIPS*, 2023.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer, 2016.

Haoyuan Qin, Chennan Ma, Mian Deng, Zhengzhu Liu, Songzhu Mei, Xinwang Liu, Cheng Wang, and Siqi Shen. The dormant neuron phenomenon in multi-agent reinforcement learning value factorization. In *NeurIPS*, 2024.

Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML*, pp. 4295–4304, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: expanding monotonic value function factorisation for deep multi-agent reinforcement learning. In *NeurIPS*, 2020.

Mikayel Samvelyan, Tabish Rashid, Christian Schröder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob N. Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *AAMAS*, pp. 2186–2188, 2019.

Siqi Shen, Mengwei Qiu, Jun Liu, Weiquan Liu, Yongquan Fu, Xinwang Liu, and Cheng Wang. Resq: A residual q function-based approach for multi-agent reinforcement learning value factorization. In *NeurIPS*, 2022.

Siqi Shen, Chennan Ma, Chao Li, Weiquan Liu, Yongquan Fu, Songzhu Mei, Xinwang Liu, and Cheng Wang. Riskq: Risk-sensitive multi-agent reinforcement learning value factorization. In *NeurIPS*, 2023. URL https://openreview.net/forum?id=FskZtRvMJI.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *ICML*, pp. 32145–32168, 2023.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Hostallero, and Yung Yi. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML*, pp. 5887–5896, 2019.

Wei-Fang Sun, Cheng-Kuang Lee, and Chun-Yi Lee. DFAC framework: Factorizing the value function via quantile mixture for multi-agent distributional q-learning. In *ICML*, pp. 9945–9954, 2021.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *AAMAS*, pp. 2085–2087, 2018.

Tristan Tomilin, Luka van den Boogaard, Samuel Garcin, Constantin Ruhdorfer, Bram Grooten, Fabrice Kusters, Yali Du, Andreas Bulling, Mykola Pechenizkiy, and Meng Fang. Meal: A benchmark for continual multi-agent reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *ICLR*, 2021.

Jiuqi Wang, Rohan Chandra, and Shangtong Zhang. Experience replay addresses loss of plasticity in continual learning. *arXiv preprint arXiv:2503.20018*, 2025.

Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *CoRR*, 2020. URL https://arxiv.org/abs/2002.03939.

Yaodong Yang, Guangyong Chen, Pheng-Ann Heng, et al. Sample-efficient multiagent reinforcement learning with reset replay. In *Forty-first international conference on machine learning*, 2024.

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. *NeurIPS*, 35:24611–24624, 2022.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. PMLR, 2017.

Appendix

A Background

A.1 Value Function Factorization

In value factorization methods Rashid et al. (2018); Sunehag et al. (2018); Wang et al. (2021); Son et al. (2019), per-agent utilities Q_i are approximated by *agent networks*, and then mixed by a *mixer network* to form the joint state-action value function Q_{tot}.

Definition A1 (Individual-Global-Max (IGM) Son et al. (2019))**.** *For a joint state-action value function $Q_{\text{jt}} : \mathcal{T}^N \times A^N \mapsto \mathbb{R}$, where $\tau \in \mathcal{T}^N$ is a joint action-observation history and $\mathbf{a} \in A^N$ is the joint action, if there exist individual state-action functions $\{Q_i : \mathcal{T}_i \times A_i \mapsto \mathbb{R}\}_{i=1}^N$ such that*

$$\arg\max_{\mathbf{a}} Q_{\text{jt}}(\tau, \mathbf{a}) = \Big(\arg\max_{a_1} Q_1(\tau_1, a_1), \ \ldots, \ \arg\max_{a_N} Q_N(\tau_N, a_N)\Big). \tag{A.1}$$

then we say that $\{Q_i\}_{i=1}^N$ satisfies IGM *for Q_{jt} under τ, or equivalently, $Q_{\text{jt}}(\tau, \mathbf{a})$ is factorized by $\{Q_i(\tau_i, a_i)\}_{i=1}^N$.*

A.2 Continual MARL Setting

Building on the Dec-POMDP notation in Section 2.1, we consider a non-stationary sequence $(\mathcal{M}_1, \ldots, \mathcal{M}_K)$ of cooperative tasks. At stage k, the active task is

$$\mathcal{M}_k = \langle \mathcal{S}_k, \mathcal{A}_k, P_k, r_k, \mathcal{O}_k, \Omega_k, N_k \rangle, \tag{A.2}$$

where $\mathcal{A}_k = \prod_{i=1}^{N_k} \mathcal{A}_{k,i}$ and $\mathcal{O}_k = \prod_{i=1}^{N_k} \mathcal{O}_{k,i}$ are the joint action and observation spaces, and N_k is the number of agents. The shared policy parameters θ are updated continuously, without reinitialization, to maximize the expected return on the active task,

$$J_k(\theta) = \mathbb{E}\left[\sum_{h=0}^{H-1} \gamma^h r_{k,h}\right]. \tag{A.3}$$

The benchmark protocols differ in how task identity is exposed. In our modified SMAC/SMACv2 and matrix-game settings, the task schedule constructs the non-stationary stream but task identity is not provided to the learner. MEAL follows its standard task-head protocol. Across all settings, KNIFE detects neurons from optimization statistics and never uses task identity or task-boundary signals to trigger intervention; it follows the same global periodic schedule. We evaluate whether the model retains sufficient plasticity to adapt efficiently to the active environment, rather than requiring simultaneous retention of high performance on all previously encountered tasks. In the scaling settings, variable agent, action, observation, and state dimensions are handled by the shared adaptive backbone described in Appendix D.1.

A.3 Knowledge Invariant Principle

Definition A2 (Knowledge Invariant Principle (KI) Qin et al. (2024))**.** *A joint state-action value function is represented as $Q_{tot}^{\theta,\phi}(\tau, \mathbf{a}) = f_\theta(Q_1^\phi(\tau_1, a_1), ..., Q_N^\phi(\tau_N, a_N))$, where f_θ is the mixing function that mixes Q_i into Q_{tot}, τ is joint observation-action history, $\mathbf{a} = [a_1, ...a_N]$ is the joint action of multi-agent, $g : \mathbb{R} \mapsto \mathbb{R}$ is a function that maps weights in θ to $\hat{\theta}$, $g(\theta) = \hat{\theta}$. $h : \mathbb{R} \mapsto \mathbb{R}$, $h(\phi) = \tilde{\phi}$, h map the weights in ϕ to $\tilde{\phi}$. If the following condition holds:*

$$\begin{aligned} Q_{\text{tot}}^{\theta,\phi}(\tau, \mathbf{a}) \geq Q_{\text{tot}}^{\theta,\phi}(\tau, \mathbf{a}') \Rightarrow Q_{\text{tot}}^{\hat{\theta},\tilde{\phi}}(\tau, \mathbf{a}) \geq Q_{\text{tot}}^{\hat{\theta},\tilde{\phi}}(\tau, \mathbf{a}'), \\ \exists!\ k :\ a_k \neq a'_k. \end{aligned} \tag{A.4}$$

then, the two functions g and h satisfy the Knowledge Invariant Principle for $Q_{\text{tot}}^{\theta,\phi}$, where $[Q_i(\tau_i, a_i)]_{i=1}^N$ is individual agent utility function, N is the number of agents, τ_i and a_i are the observation-action history and action of agent i, respectively. $\exists!$ represents the concept of unique existence.

The KI principle is used to assess whether a parameter perturbation operation preserves the learned cooperative knowledge. Qin et al. (2024) show that if a parameter perturbation violates the KI principle, then it breaks the IGM consistency and therefore cannot guarantee IGM-consistent value factorization.

A.4 KNOWLEDGE INVARIANT++ (KI++) PRINCIPLE

Inspired by Qin et al. (2024), we propose a stronger neuron-level parameter perturbation principle, Knowledge Invariant++ Principle. This principle removes the unique existence condition $\exists!\ k:\ a_k \neq a'_k$ from the KI principle.

Definition A3 (Knowledge Invariant++ Principle (KI++))**.** *A joint state-action value function is represented as $Q_{tot}^{\theta,\phi}(\tau, \mathbf{a}) = f_\theta(Q_1^\phi(\tau_1, a_1), ..., Q_N^\phi(\tau_N, a_N))$, where f_θ is the mixing function that mixes Q_i into Q_{tot}, τ is joint observation-action history, $\mathbf{a} = [a_1, ...a_N]$ is the joint action of multi-agent, $g : \mathbb{R} \mapsto \mathbb{R}$ is a function that maps weights in θ to $\hat{\theta}$, $g(\theta) = \hat{\theta}$. $h : \mathbb{R} \mapsto \mathbb{R}$, $h(\phi) = \tilde{\phi}$, h map the weights in ϕ to $\tilde{\phi}$. If the following condition holds:*

$$Q_{\text{tot}}^{\theta,\phi}(\tau, \mathbf{a}) \geq Q_{\text{tot}}^{\theta,\phi}(\tau, \mathbf{a}') \Rightarrow Q_{\text{tot}}^{\hat{\theta},\tilde{\phi}}(\tau, \mathbf{a}) \geq Q_{\text{tot}}^{\hat{\theta},\tilde{\phi}}(\tau, \mathbf{a}'), \tag{A.5}$$

then, the two functions g and h satisfy the Knowledge Invariant++ Principle for $Q_{\text{tot}}^{\theta,\phi}$, where $[Q_i(\tau_i, a_i)]_{i=1}^N$ is individual agent utility function, N is the number of agents, τ_i and a_i are the observation-action history and action of agent i, respectively.

The unique existence condition $\exists!\ k:\ a_k \neq a'_k$ from the KI principle states that two joint actions must differ in exactly one agent's action. This requirement, however, is overly strict. It is specifically designed for monotonic increasing mixing functions, where the value comparison between two joint actions relies on the change in only one agent's action. The KI++ principle is stronger than the KI principle in the sense that it is less restrictive and therefore more widely applicable.

A.5 DIFFERENCE BETWEEN REBORN AND KNIFE

ReBorn is a neuron-level method designed to address dormant neurons in MARL and provides a KI guarantee for algorithms such as QMIX, QPLEX, and RMIX. In contrast, KNIFE targets stagnant and volatile neurons and provides the stronger KI++ guarantee for its injection operation under the architectural conditions stated in Theorem 1. Empirically, KNIFE outperforms ReBorn across multiple environments, including SMACv2, Predator–Prey, and the one-step matrix game.

A.6 DETAILED COMPARISON WITH PLASTICITY INJECTION (PI)

Plasticity Injection (PI) and KNIFE both introduce new neurons and preserve learned knowledge via frozen weights. However, PI operates at the layer level, targeting the last K layers of the network, whereas KNIFE acts at the neuron level, focusing specifically on stagnant and volatile neurons in the mixing network. They differ in when, where, and how plasticity is injected, as detailed in Table A1.

A.6.1 DETAILED COMPARISON OF INITIALIZATION STRATEGIES

Plasticity Injection (PI) and KNIFE use related function-preserving-at-injection fresh/frozen capacity constructions, but differ in the granularity and allocation of plastic capacity. PI introduces capacity at the layer level, whereas KNIFE allocates it locally to detected neurons and uses a state-aware initialization for the active branch.

Plasticity Injection (PI) Initialization: PI typically employs standard Kaiming initialization for both the weights and biases of the injected units. This initialization is *state-agnostic*: it does not condition the newly introduced parameters on the current optimization state or weight distribution of the surrounding network. In some settings, the resulting units may enter activation dead zones (e.g., dying ReLUs) or initially propagate weak gradient signals.

KNIFE State-Aware Initialization: To encourage newly injected active neurons to contribute early in training and reduce dead-zone risk, KNIFE applies a *state-aware* initialization strategy tailored to the current network conditions:

Table A1: Comparison of PI and KNIFE.

Feature	Plasticity Injection (PI)	KNIFE (Ours)	Implication
Granularity	Layer-level	Neuron-level	Operates at neuron-level granularity.
Trigger Time	Once	Periodically	Supports repeated intervention during training.
Treatment Location	All neurons of the last K layers	Stagnant/volatile neurons in the mixing network	Allocates additional capacity only to selected neurons.
Increased Neurons	$K \times N \times 2$ where N is the average number of neurons in the last K layers	$n \times 2$, where n is the number of stagnant/volatile neurons	Additional capacity scales with the number of selected neurons.
Recycling Mechanism	No	Frozen neurons are recycled through pruning.	Recycles frozen branches after decay.
Initialization	Kaiming Initialization (State Agnostic)	State-Aware	Uses network-state information to reduce dead-zone risk.

- **Positive Input Bias:** Unlike PI, which typically initializes biases using a zero-centered distribution, KNIFE explicitly initializes the input bias of the active neuron to a positive value (e.g., proportional to the standard deviation of its input weights). This shift is designed to increase the likelihood of positive-valued activations and thereby facilitate early gradient updates.
- **Contextual Output Weights:** The active neuron's output weights are initialized using a zero-centered Gaussian distribution whose variance is dynamically tied to the standard deviation of the subsequent layer's weights (safeguarded by a minimum threshold). This choice is intended to align the initial forward-signal scale with the existing network structure.
- **Input Weights:** Standard Kaiming initialization is retained solely for the input weights to maintain proper variance scaling.

The state-aware initialization is intended to support gradient flow and reduce the likelihood that an active branch immediately enters an activation dead zone. This design differs from PI in the granularity and allocation of plastic capacity rather than in the general use of fresh/frozen branches that preserve the function at injection. Although PI can also satisfy KI++ by preserving the network function at injection, KI++ alone does not specify where plasticity should be introduced: PI applies layer-level injection, whereas KNIFE explicitly identifies and repairs plasticity-impaired neurons.

B Principle and Theorem

Theorem 1. *Consider a value-factorization model in which KNIFE is applied only to mixing-network layers whose neuron contributions are combined through affine transformations and pointwise activations. Immediately after KNIFE injection, before optimization or decay, the state-action value function* $Q_{tot}^{\theta,\phi}(e\tau, ea)$ *satisfies the KI++ principle.*

Scope of the guarantee. This guarantee concerns the intervention boundary only. Subsequent optimization of the active branch and decay of the frozen branches are intentionally allowed to change the value function so that the model can adapt to the current task. Thus, KI++ prevents an abrupt function change caused by injection itself; it is not intended to constrain the subsequent adaptation process.

$$Q_{\text{tot}}^{\theta,\phi}(\tau, \boldsymbol{a}) = f_\theta\Big(\tau, Q_1^\phi(\tau_1, a_1), \ldots, Q_N^\phi(\tau_N, a_N)\Big). \tag{B.6}$$

where $Q_{tot}^{\theta,\phi}(\boldsymbol{e}\tau, \boldsymbol{e}a)$ is the joint state-action value function and f_θ is the mixing function of a supported value-factorization architecture (e.g., QMIX Rashid et al. (2018), QPLEX Wang et al. (2021), or DMIX Sun et al. (2021)).

In Theorem 1, KNIFE is applied only to the mixing network. Therefore, the agent-network parameters remain unchanged, i.e.,

$$h(\phi) = \phi,$$

and no structural or linearity assumption on h is required. It therefore suffices to analyze the KNIFE mapping $g : \theta \mapsto \hat{\theta}$ and show that the mixing function is unchanged immediately after injection.

Consider any neuron i in layer l of the injected mixing network. By construction, each pre-injection neuron is expanded into three injected neurons (knowledge / active / compensation), and satisfies:

$$x_i^{l,\text{know}} + x_i^{l,\text{active}} + x_i^{l,\text{comp}} = x_i^{l,\text{know}} = x_i^l. \tag{B.7}$$

where x_i^l denotes the pre-injection activation of neuron i at layer l.

Therefore, for any layer l, the layer activation vector is unchanged:

$$x_{\text{after}}^l = x_{\text{before}}^l. \tag{B.8}$$

Consequently, for the whole network, for any inputs τ and $\boldsymbol{q}$,

$$f_{\hat{\theta}}(\tau, \boldsymbol{q}) = f_\theta(\tau, \boldsymbol{q}). \tag{B.9}$$

Proof.

$$Q_{tot}^{\theta,\phi}(\tau, \boldsymbol{a}) \geq Q_{tot}^{\theta,\phi}(\tau, \boldsymbol{a}'), \tag{B.10}$$

$$Q_{tot}^{\theta,\phi}(\tau, [a_1, \ldots, a_N]) \geq Q_{tot}^{\theta,\phi}(\tau, [a'_1, \ldots, a'_N]), \quad \text{expand } \boldsymbol{a}. \tag{B.11}$$

$$f_\theta\Big(\tau, Q_1^\phi(\tau_1, a_1), \ldots, Q_N^\phi(\tau_N, a_N)\Big) \geq f_\theta\Big(\tau, Q_1^\phi(\tau_1, a'_1), \ldots, Q_N^\phi(\tau_N, a'_N)\Big). \tag{B.12}$$

$$f_{\hat{\theta}}\Big(\tau, Q_1^\phi(\tau_1, a_1), \ldots, Q_N^\phi(\tau_N, a_N)\Big) = f_\theta\Big(\tau, Q_1^\phi(\tau_1, a_1), \ldots, Q_N^\phi(\tau_N, a_N)\Big) \quad \text{because of (B.9).} \tag{B.13}$$

$$f_{\hat{\theta}}\Big(\tau, Q_1^\phi(\tau_1, a'_1), \ldots, Q_N^\phi(\tau_N, a'_N)\Big) = f_\theta\Big(\tau, Q_1^\phi(\tau_1, a'_1), \ldots, Q_N^\phi(\tau_N, a'_N)\Big) \quad \text{because of (B.9).} \tag{B.14}$$

$$f_{\hat{\theta}}\Big(\tau, Q_1^\phi(\tau_1, a_1), \ldots, Q_N^\phi(\tau_N, a_N)\Big) \geq f_{\hat{\theta}}\Big(\tau, Q_1^\phi(\tau_1, a'_1), \ldots, Q_N^\phi(\tau_N, a'_N)\Big). \tag{B.15}$$

$$Q_{tot}^{\hat{\theta},\phi}(\tau, \boldsymbol{a}) \geq Q_{tot}^{\hat{\theta},\phi}(\tau, \boldsymbol{a}'). \tag{B.16}$$

Since $h(\phi) = \phi$ and $f_{\hat{\theta}}(\tau, \mathbf{q}) = f_\theta(\tau, \mathbf{q})$ for every input immediately after injection, the ordering between any two joint actions is preserved. Therefore, the state-action value function satisfies the KI++ principle for $Q_{\text{tot}}^{\theta,\phi}$. □

Stagnant neurons in $\mathcal{S}_{stag}$ exhibit disproportionately small gradient-based update activity relative to their parameter scale and layer-wise peers, resulting in excessive optimization inertia and inefficient adaptation.

Volatile neurons in S_{vol} lie in the high-RUA tail of the layer-wise update-activity distribution and undergo disproportionately large relative updates compared with their peers. We treat them as a complementary, rather than primary, repair target. Empirically, leaving volatile neurons untreated is associated with higher subsequent stagnant-neuron ratios and weaker long-term adaptation, while jointly repairing stagnant and volatile neurons provides an additional performance gain.

C ALGORITHM

The KNIFE algorithm is described in Algorithm 1. At each learner update, RUA uses gradients recorded after backpropagation and before the optimizer step; these per-update statistics are accumulated over interval T.

Algorithm 1 KNIFE: Knowledge-retentive Neuron-level Plasticity Focusing Injection

Require: network parameters Θ; intervention interval $T = 0.2$M environment steps; thresholds (α, β); per-learner-update frozen-branch decay factor $\gamma \in (0, 1)$; prune threshold $\varepsilon_{\text{prune}} = 10^{-4}$

Initialize frozen-neuron registry $\mathcal{F} \leftarrow \emptyset$
Initialize composite-unit registry $\mathcal{C} \leftarrow \emptyset$
Initialize the global environment-step clock $c \leftarrow 0$ and next intervention time $c_{\text{next}} \leftarrow T$ at the beginning of Task 0
for each learner update $k = 1, 2, \ldots$ **do**
  Set c to the current total number of environment steps
  Record pre-optimizer UA statistics and update Θ by gradient descent **only** on trainable parameters
  if $c \geq c_{\text{next}}$ **then**
    for each layer ℓ to be intervened **do**
      Compute RUA scores from accumulated UA statistics for ordinary neurons not registered in $\mathcal{C}$
      $\mathcal{S}^{\ell}_{\text{stag}} \leftarrow \{i \mid \text{RUA}^{\ell}_{i} < \alpha\}$
      $\mathcal{S}^{\ell}_{\text{vol}} \leftarrow \{i \mid \text{RUA}^{\ell}_{i} > \beta\}$
      $\mathcal{S}^{\ell}_{\text{repair}} \leftarrow \mathcal{S}^{\ell}_{\text{stag}} \cup \mathcal{S}^{\ell}_{\text{vol}}$
      for each neuron $i \in \mathcal{S}^{\ell}_{\text{repair}}$ **do**
        (Injection) Create three parallel branches: know (frozen), active (trainable), comp (frozen)
        Copy original params to knowledge branch:
          $\mathbf{w}^{\ell,\text{know}}_{in,i} \leftarrow \mathbf{w}^{\ell}_{in,i}$, $b^{\ell,\text{know}}_{i} \leftarrow b^{\ell}_{i}$, $\mathbf{w}^{\ell,\text{know}}_{out,i} \leftarrow \mathbf{w}^{\ell}_{out,i}$
        Initialize active branch (e.g., Kaiming/He init):
          $\mathbf{w}^{\ell,\text{active}}_{in,i} \sim \mathcal{N}(0, \delta^2_{in})$, $b^{\ell,\text{active}}_{i} \leftarrow 0.5\delta_{in}$, $\mathbf{w}^{\ell,\text{active}}_{out,i} \sim \mathcal{N}(0, \delta^2_{out})$
        Set compensation branch to cancel the active branch at injection:
          $\mathbf{w}^{\ell,\text{comp}}_{in,i} \leftarrow \mathbf{w}^{\ell,\text{active}}_{in,i}$, $b^{\ell,\text{comp}}_{i} \leftarrow b^{\ell,\text{active}}_{i}$, $\mathbf{w}^{\ell,\text{comp}}_{out,i} \leftarrow -\mathbf{w}^{\ell,\text{active}}_{out,i}$
        Freeze parameters of know/comp branches; keep active trainable
        Add know/comp to frozen registry: $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\ell, i, \text{know}), (\ell, i, \text{comp})\}$
        Register the three branches as one composite unit in $\mathcal{C}$
      end for
    end for
    $c_{\text{next}} \leftarrow c_{\text{next}} + T$
  end if
  for each frozen branch $f \in \mathcal{F}$ **do**
    Let $\theta_f \leftarrow \text{vec}(\mathbf{w}_{in,f}, b_f, \mathbf{w}_{out,f})$
    (Per-update decay) $\theta_f \leftarrow \gamma\, \theta_f$
    if $\|\theta_f\|_2 < \varepsilon_{\text{prune}}$ **then**
      Remove/prune branch f from the network and from $\mathcal{F}$
    end if
  end for
  Remove a unit from $\mathcal{C}$ once both of its frozen branches have been pruned
end for

D EXPERIMENTAL DETAILS

D.1 EXPERIMENTAL SETUP

We implement these algorithms based on their open-source repositories to carry out performance analyses with hyperparameters consistent with those in PyMARL. Our methods are implemented

Table A2: Common hyperparameters used in our experiments.

Hyperparameter	SMAC	SMACv2	One-step Matrix Game	Predator-Prey
Action Selector	epsilon greedy	epsilon greedy	epsilon greedy	epsilon greedy
Batch Size	32	32	32	32
Buffer Size	5000	5000	5000	5000
Learning Rate	0.0005	0.0005	0.0005	0.0005
Hypernet Embed Dimension	64	64	4	64
Target Update Interval	200	200	200	200
KNIFE Intervention Interval	0.2M	0.2M	0.2M	0.2M
Stagnant Threshold α	0.25	0.25	0.25	0.25
Volatile Threshold β	3	3	3	3
Frozen-branch decay factor γ (per learner update)	0.995	0.995	0.995	0.995

Table A3: Method-specific configurations and intervention schedules.

Method	Method-specific configuration	Intervention schedule
ReDo	Dormant threshold $\tau = 0.1$	Every 0.2M steps
PI	Final two layers of the mixing network	Once at 25% of the entire training horizon
ReBorn	Dormant threshold $\alpha = 0.1$, overactive threshold $\beta = 3$	Every 0.2M steps
ReGraMa	Normalized-gradient threshold $\tau_g = 0.01$	Every 0.2M steps
CBP	Replacement rate $\rho = 10^{-4}$, utility EMA decay $\eta = 0.99$, maturity threshold $m = 100$	Every learner update
S&P	Shrinkage coefficient $\lambda = 0.995$, perturbation scale $\sigma = 10^{-4}$	Every 0.2M steps
ReSet	Last mixing-network layer, Xavier reinitialization	Every 0.2M steps
KNIFE	$\alpha = 0.25$, $\beta = 3$, $\gamma = 0.995$ after every learner update	Every 0.2M steps from the start of training

within the PyMARL framework. All reported performance results use at least five independent random seeds and report 95% confidence intervals. Unless otherwise specified, experiments use five random seeds; experiments using more seeds are explicitly identified in the corresponding sections. Additional higher-seed robustness evaluations are provided in Appendix D.13. Specific hyperparameters of different environments are listed in Table 2. We conduct experiments on a cluster equipped with multiple NVIDIA GeForce RTX 4090 GPUs. KNIFE uses a fixed identification-and-injection interval of $T = 0.2$M environment steps in every environment. The global periodic schedule starts at the beginning of Task 0 and does not reset at task boundaries. Separately, the frozen knowledge and compensation parameters are multiplied by the decay factor γ after every learner update, followed by a pruning-threshold check.

Baseline configurations. All methods share the same backbone architecture, optimizer, learning rate, batch size, replay-buffer configuration, and training budget. For the compared plasticity methods, we follow their default method-specific configurations. KNIFE uses a fixed intervention configuration across the main environments without task-specific or environment-specific tuning. The method-specific hyperparameters and intervention schedules are summarized in Table A3.

Fairness and tuning policy. All periodic intervention methods use the same 0.2M-step execution interval from the beginning of training, while method-specific thresholds and update rules follow their published default configurations. PI retains its native one-shot schedule and CBP retains its per-update replacement rule because these schedules are integral to their respective algorithms. We do not perform environment-specific or task-specific tuning for either KNIFE or the baselines. Accordingly, the reported comparisons evaluate standardized implementations under a common tuning policy rather than the best attainable performance after exhaustive environment-specific tuning.

Benchmarks. We modify existing MARL benchmarks (SMAC, SMACv2, Predator Prey) for four continual learning setting: task-drift, task-switch, scaling up and scaling down. Moreover, we also evaluate non-continual learning settings. We describe the settings as follows.

(1) **The task-drift setting** comprises 11 SMACv2 tasks. Each team has 5 agents consisting two different type whose probability changes every task. (2) **The task-switch setting** switches 2 tasks periodically. For example, in the SMACV2 5 protoss scenario, the tasks switch between 5 Stalker vs 5 Stalker and 5 Zealot vs 5 Zealot. Predator Prey is evaluated under similar setting. The first two settings can be viewed in the Table A4 in detail.(3) **The scaling up setting** comprise 3 SMAC tasks where the number of agents and the opponents increase from 3, to 5, and 8. (4) **The scaling**

Table A4: Modified SMACv2 task-switch and task-drift configurations.

Race	Unit order	Task Switch (2 phases)	Task Drift (11 phases)
Protoss	[Stalker, Zealot, Colossus]	[1.0, 0.0, 0.0] ↔ [0.0, 1.0, 0.0]	[1.0,0.0,0.0] → [0.9,0.1,0.0] → ⋯ → [0.0,1.0,0.0]
Terran	[Marine, Marauder, Medivac]	[1.0, 0.0, 0.0] ↔ [0.0, 1.0, 0.0]	[1.0,0.0,0.0] → [0.9,0.1,0.0] → ⋯ → [0.0,1.0,0.0]
Zerg	[Zergling, Baneling, Hydralisk]	[1.0, 0.0, 0.0] ↔ [0.0, 0.0, 1.0]	[1.0,0.0,0.0] → [0.9,0.0,0.1] → ⋯ → [0.0,0.0,1.0]

down setting evaluate 3 SMAC tasks that 5 agents fight against 6, 5, and 4 enemies respectively. (5) **Non-continue setting** evaluate the original SMAC and SMACv2.

For the original **SMAC**, we evaluate *structural shifts* where the state and action spaces change due to team size variations every 2×10^6 environment steps. This includes **agent scaling** where the team size increases (e.g., 3m → 5m → 8m), and **opponent variation** where the number of enemies fluctuates across tasks (e.g., 5m_vs_6m → 5m → 5m_vs_4m).

Implementation for scaling-up/down settings. For the scaling-up and scaling-down experiments, all compared methods use the same adaptive backbone to accommodate changes in the numbers of agents, actions, and input dimensions across tasks. Variable-length local observations and global states are first mapped to fixed-dimensional representations by shared encoders, such that the subsequent agent and mixing networks maintain fixed input dimensions throughout training. The agent network is shared across agents, while the action head and the mixing network use pre-allocated output slots with capacities covering the largest task configuration in the sequence. At each task, only the slots corresponding to the currently available agents and actions are used. Therefore, switching between tasks does not resize or re-initialize the base MARL network. This adaptation mechanism is identical for all compared methods and is independent of the neuron-level plasticity injection introduced by KNIFE.

SMACv2. We modify **SMACv2** to evaluate *compositional shifts* with two controlled settings: **Task Switch** and **Task Drift**. In **Task Switch**, the scenario alternates between two mirror-match tasks every 5×10^5 environment steps. For example, in 5gen_protoss, the task switches between 5 Stalker vs 5 Stalker and 5 Zealot vs 5 Zealot. We apply the same pattern to other races (Terran and Zerg) by switching between two unit types. In **Task Drift**, the unit composition changes gradually in discrete stages: every 5×10^5 steps, the spawn probability of the first unit type decreases by 0.1 (from 1.0 to 0.0), while that of the second increases by 0.1 (from 0.0 to 1.0), producing 10 drift updates and thus 11 task stages in total.

One-step Matrix Game (Sequential Setting). In the one-step matrix game, two agents must learn to coordinate their actions to maximize a shared global reward. Each agent has a discrete action space of size 3, resulting in a 3×3 joint payoff matrix. The initial Task 1 stage and the intervening Tasks 2–5 are each trained for 5×10^5 environment steps. Only the final revisit to Task 1 is doubled to 10^6 steps; the "w/o transfer" comparison receives the same 10^6-step target-task budget. We record ten evenly spaced recovery measurements during this final revisit. To create a challenging task transfer scenario that rigorously tests the plasticity of the mixing network, we designed five distinct payoff matrices where the optimal joint action shifts drastically.

The base matrix, denoted as M_0, features a relatively smooth reward gradient guiding the agents towards the optimal joint action at $(2, 2)$ with a maximum reward of 4:

$$M_0 = \begin{bmatrix} -6 & -4 & -2 \\ -3 & -1 & 1 \\ 0 & 2 & 4 \end{bmatrix} \tag{D.17}$$

To explicitly evaluate the agents' adaptability and to induce optimization inertia, we introduce four additional tasks (M_1 to M_4). In these matrices, the global optimum (with a high reward of 12) is sharply located at different coordinates, surrounded by severe penalties (large negative rewards). This sparse and deceptive reward landscape forces the network to completely overwrite its previous coordination conventions when the task switches:

$$M_1 = \begin{bmatrix} -10 & -10 & 12 \\ -9 & -12 & -8 \\ -4 & -8 & -11 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -11 & -8 & -4 \\ -8 & -12 & -9 \\ 12 & -10 & -10 \end{bmatrix} \tag{D.18}$$

$$M_3 = \begin{bmatrix} -11 & -4 & -11 \\ -4 & 12 & -4 \\ -11 & -4 & -11 \end{bmatrix}, \quad M_4 = \begin{bmatrix} 12 & -8 & -10 \\ -8 & -12 & -6 \\ -10 & -6 & -11 \end{bmatrix} \tag{D.19}$$

Specifically, the optimal joint actions (a_1, a_2) for M_1, M_2, M_3, and M_4 are located at $(0, 2)$, $(2, 0)$, $(1, 1)$, and $(0, 0)$, respectively. This extreme variation across tasks creates a challenging setting in which neurons with low relative update activity can hinder adaptation to subsequent tasks.

One-step Matrix Game (Cyclic Setting). Unlike the sequential setup evaluated in the main text, each unique payoff matrix constitutes a task. The agents must learn a sequence of four tasks (Task 1–4), with the active payoff matrix switching every 5×10^5 environment steps, as illustrated in the left panel of Figure A10.

Predator–Prey. Predator–Prey is a grid-world cooperation task where predators must coordinate to capture preys; a capture succeeds only when at least two predators execute the capture action simultaneously. We construct a non-stationary task stream by periodically switching the reward profile between summer and winter. In both phases, the time reward is 0 and capturing a hare yields $+5$. The only change is the stag reward: it is $+10$ in summer but -5 in winter. Thus, the task shift purely comes from a sign flip in the incentive for stag captures, while the environment dynamics remain unchanged.

Baselines. We compare against neuron-level plasticity injection baselines including ReDo Sokar et al. (2023), ReBorn Qin et al. (2024), ReGraMa Liu et al. (2025b), S&P Ash & Adams (2020), CBP Dohare et al. (2024), ReSet Nikishin et al. (2022), and Plasticity Injection (PI) Nikishin et al. (2023). Note that although ReSet can reduce the stagnant ratio, its large-scale resetting may forget previously learned knowledge, leading to slow convergence and poor overall performance under task changes.

D.2 Stagnant Neuron in MARL

The Plasticity Loss with New MARL Tasks In our modified SMACv2 scenarios, two opposing teams of five agents engage in combat. To explicitly evaluate sequential adaptation, we change the unit composition for *both* sides according to a fixed task sequence:

- **Protoss Setting:** 100% Stalkers → 100% Zealots → 100% Colossi → 100% Stalkers.
- **Terran Setting:** 100% Marines → 100% Marauders → 80% Marines + 20% Medivacs → 100% Marines.

Figure A1 illustrates the adaptation dynamics of standard value factorization methods (QMIX, QPLEX, and DMIX) under these sequential task changes. In both streams, Task 1 is trained during 0–1M steps, Tasks 2 and 3 during 1M–1.5M and 1.5M–2M steps, respectively, and Task 1 is revisited during 2M–4M steps. The w/o-transfer models are trained only on Task 1. In panels (a) and (c), their horizontal dashed segments during 1M–2M are visual placeholders indicating that these models receive no updates while the continual models train on Tasks 2 and 3; they do not represent evaluations on those tasks.

Panels (b) and (d) isolate re-adaptation to Task 1. The solid curves are the continual models' final Task 1 interval from 2M–4M, shifted to a 0–2M horizontal axis, whereas the dashed curves are the w/o-transfer models trained on Task 1 from 0–2M. Under this aligned target-task window, QMIX, QPLEX, and DMIX re-adapt more slowly and reach lower performance than their w/o-transfer counterparts in both settings. These results provide consistent evidence of impaired re-adaptation after sequential task transfers.

More examples of plasticity loss in MARL when facing new tasks are shown in the Figure A2. After task switching, the neural network suffers a loss of plasticity, which leads to performance degradation.

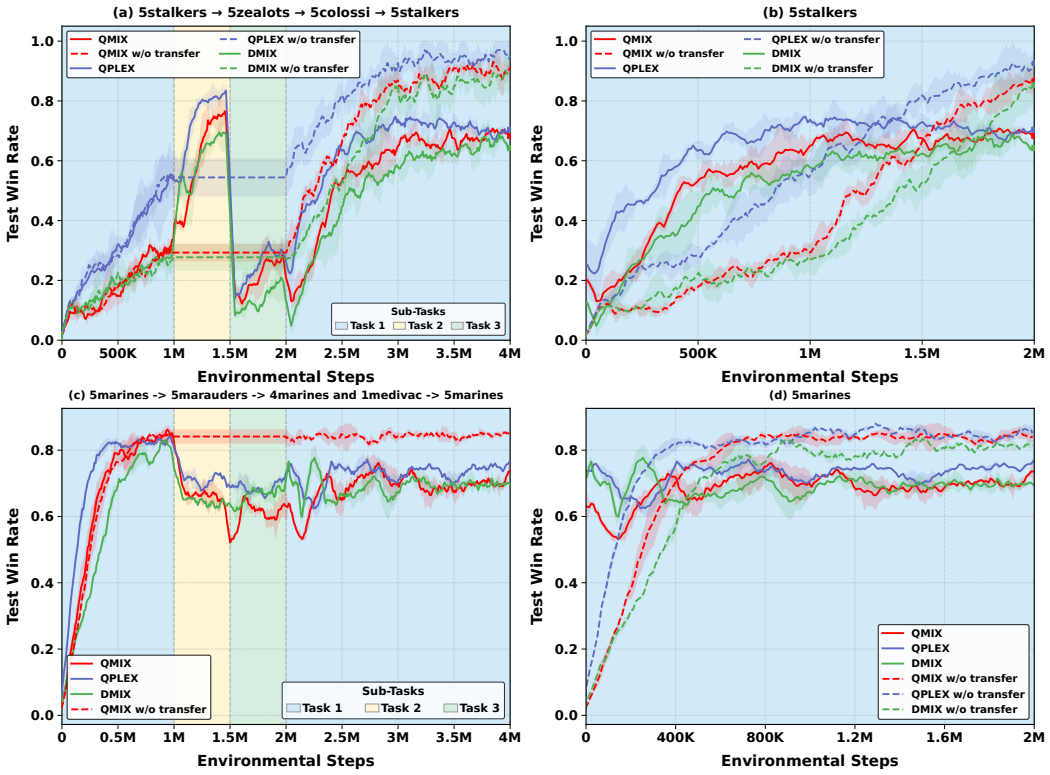

Figure A1: Plasticity loss in modified SMACv2 with sequential tasks. (a,c) Complete Protoss and Terran streams; horizontal dashed segments mark the interval in which the w/o-transfer models receive no training because they are not exposed to Tasks 2 and 3. (b,d) Re-adaptation to the initial task, comparing the continual models' 2M–4M revisit (shifted to 0–2M) with w/o-transfer single-task training from 0–2M.

Table A5: Matrix bias across the initial nine-task repair-ratio sequence.

Repair ratio	T0	T1	T2	T3	T4	T5	T6	T7	T8
0%	0.52 ± 0.18	16.96 ± 2.63	17.65 ± 2.48	1.59 ± 0.35	13.27 ± 0.23	23.66 ± 4.40	38.95 ± 1.10	21.13 ± 4.31	57.26 ± 2.19
25%	0.26 ± 0.06	16.80 ± 2.67	23.68 ± 3.54	1.13 ± 0.26	17.57 ± 3.76	32.25 ± 3.56	35.74 ± 1.08	35.67 ± 3.93	58.27 ± 2.36
50%	0.26 ± 0.05	16.74 ± 2.68	20.55 ± 3.21	0.79 ± 0.09	12.75 ± 0.23	27.86 ± 4.48	33.40 ± 3.26	34.87 ± 3.87	53.67 ± 4.12
75%	0.46 ± 0.17	16.94 ± 2.65	17.36 ± 2.51	1.00 ± 0.19	12.73 ± 0.14	22.13 ± 4.13	31.74 ± 3.71	32.16 ± 3.14	50.95 ± 3.96
100%	0.77 ± 0.23	16.82 ± 2.67	17.60 ± 2.49	1.59 ± 0.37	13.66 ± 0.34	24.31 ± 4.77	35.97 ± 0.85	28.92 ± 4.39	47.15 ± 4.44

Repairing naturally occurring stagnant neurons. We run a separate dose–response study over a nine-task matrix sequence (Tasks 0–8), with 5×10^5 environment steps per stage, followed by eight recovery stages on Task 0. During recovery, we repair a fixed fraction of the naturally detected stagnant neurons while leaving all other training settings unchanged. Tables A5 and A6 report mean $\pm$ 95% confidence interval over 10 random seeds. R1–R8 denote the recovery checkpoints, and "Mean" is the aggregate recovery metric used in Figure 3(c). Lower matrix bias is better.

Expressiveness study: injecting stagnant/volatile neurons. To study how stagnant neurons affect the expressive power of value factorization networks, we create controlled stagnant/volatile neurons by directly manipulating gradients. Specifically, we select a subset of neurons in the mixer network and modify their backpropagated gradients during training. For **stagnant neurons**, we freeze learning by masking their gradients to zero (i.e., gradient freezing). For **volatile neurons**, we amplify their gradients by a factor of 100 before the optimizer update. This intervention changes the effective update magnitude (and thus the RUA) without changing the forward computation graph. The detailed results are shown in Figure A3. Increasing the number of volatile neurons is accompanied by a higher stagnant-neuron ratio and worse performance, suggesting an interaction between the two RUA tails without establishing volatile neurons as an upstream cause of degradation.

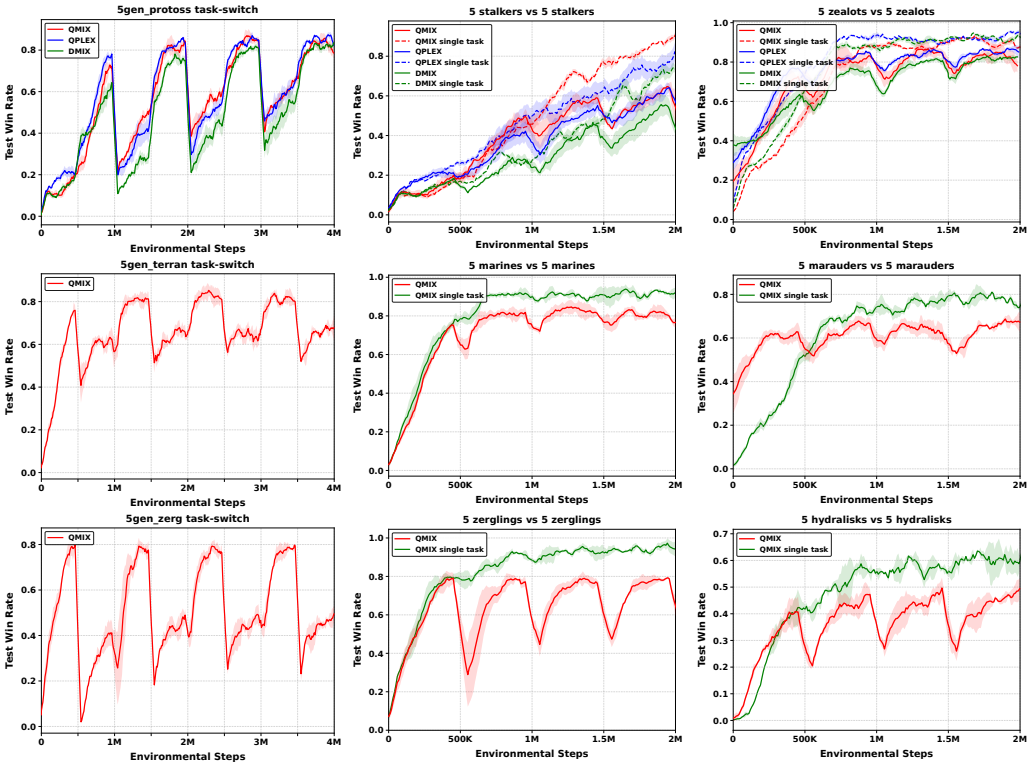

Figure A2: The plasticity loss with new MARL tasks

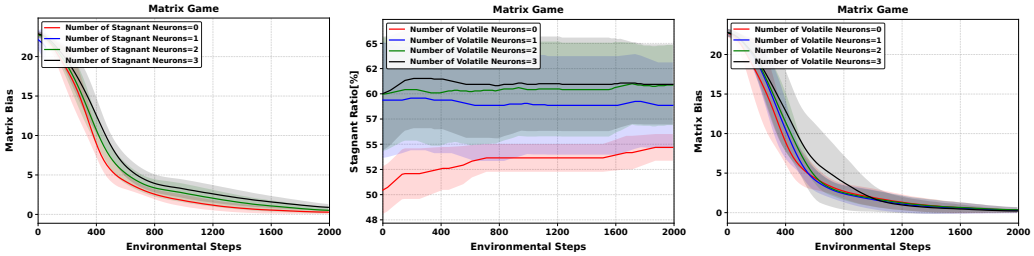

Figure A3: The influence of stagnant and volatile neurons

Weight–gradient composition of stagnant neurons. To examine whether stagnant neurons are primarily characterized by large parameter magnitudes, we further partition the detected stagnant neurons according to their gradient magnitude and parameter scale. A neuron is assigned to the small-gradient group when its GraMA score is below 0.1 and to the non-small-gradient group otherwise. Neurons with weight norms above the layer-wise median are categorized as high-weight neurons, and the remaining ones as low-weight neurons. The resulting composition is reported in Table A7 as percentages of detected stagnant neurons.

Overall, 55.9% of stagnant neurons belong to the low-weight group, whereas 44.1% belong to the high-weight group. Moreover, stagnant neurons occur in both the small- and non-small-gradient groups. These results show that excessively large absolute weights are neither necessary nor dominant for neuron stagnation. Instead, stagnation is characterized by disproportionately small gradient-based update activity relative to the neuron's parameter scale and its layer-wise peers, which can arise from small gradients, large parameter magnitudes, or their interaction.

Table A8 compares full KNIFE with stagnant-only repair. Win rate is higher-is-better, whereas stagnant and volatile ratios are lower-is-better.

Table A6: Task-0 recovery bias under different stagnant-neuron repair ratios.

Repair ratio	R1	R2	R3	R4	R5	R6	R7	R8	Mean
0%	18.73 ± 2.96	18.48 ± 2.99	18.79 ± 3.01	18.22 ± 2.95	14.95 ± 3.71	15.12 ± 3.80	14.99 ± 3.79	15.19 ± 3.84	18.33 ± 2.86
25%	18.52 ± 3.01	16.88 ± 2.95	15.33 ± 3.55	15.18 ± 3.65	15.03 ± 3.69	15.04 ± 3.66	15.03 ± 3.67	14.95 ± 3.73	17.02 ± 2.92
50%	10.29 ± 3.55	7.76 ± 3.74	7.78 ± 3.73	8.13 ± 3.84	7.60 ± 3.76	7.64 ± 3.80	7.87 ± 3.85	7.56 ± 3.82	10.12 ± 3.42
75%	8.89 ± 3.22	5.67 ± 3.73	5.60 ± 3.74	5.80 ± 3.73	5.55 ± 3.77	5.54 ± 3.83	5.67 ± 3.84	5.49 ± 3.78	7.22 ± 3.37
100%	4.44 ± 2.91	3.05 ± 1.84	0.63 ± 0.21	0.33 ± 0.10	0.38 ± 0.13	0.55 ± 0.20	0.56 ± 0.17	0.58 ± 0.17	3.94 ± 0.82

Table A7: Gradient–weight composition of detected stagnant neurons (%).

Weight Norm	Small Gradient	Non-small Gradient
Low Weight	$53.6 \pm 3.2\%$	$2.3 \pm 1.2\%$
High Weight	$34.8 \pm 2.1\%$	$9.3 \pm 1.8\%$

Training Dynamics of Stagnant and Volatile Neurons From Task 2 onward, full KNIFE consistently achieves a higher mean win rate than KNIFE (Stagnant), indicating that volatile-neuron repair provides an additional benefit beyond stagnant-neuron repair alone. More importantly, their long-term neuron dynamics differ substantially. Full KNIFE maintains a relatively stable stagnant-neuron ratio throughout the task sequence, whereas the stagnant ratio of KNIFE (Stagnant) gradually increases, reaching 47.57% at Task 11.

A similar pattern is observed for volatile neurons. Without directly repairing volatile neurons, the volatile-neuron ratio of KNIFE (Stagnant) increases from 15.03% at Task 1 to 21.22% at Task 11 and exceeds QMIX during the final tasks. In contrast, full KNIFE maintains substantially lower volatile-neuron ratios and reaches 10.28% at Task 11. Together, the increasing stagnant and volatile ratios under KNIFE (Stagnant), along with its persistent performance gap from full KNIFE, suggest that leaving volatile neurons untreated is associated with renewed neuron stagnation and weaker long-term adaptation.

The Distribution of Stagnant Neurons We find that stagnant neurons concentrate in the mixing network in the Figure A4.

Target Update Interval and Task Update Interval. We investigate how **target non-stationary** amplifies stagnation. We consider two practical sources: *implicit drift* from TD bootstrapping (controlled by the target-network update interval) and *explicit shifts* from task switching (controlled by the switching period). Figure A5 (a) shows the stagnant ratio under different target-network update intervals: an intermediate interval yields the lowest stagnant ratio (200 steps), followed by 50, whereas very frequent updates (10) are associated with substantially higher stagnation. Thus, even without task switches, the implicit non-stationarity introduced by TD learning co-occurs with the accumulation of low-plasticity units. In Figure A5 (b), more frequent task switching is accompanied by faster growth and a higher stagnant ratio, showing that explicit task changes are another condition under which stagnation becomes more pronounced.

We study whether the detected stagnant/volatile neurons persist across time. To this end, at each iteration (every time interval), we calculate the overlap coefficient between the current and previous iterations for stagnant/volatile neurons. As shown in Figure A6, there are large overlaps between stagnant neurons across iterations, as well as the volatile neurons. This indicates that once a neuron becomes stagnant/volatile, it will remain the same type of neuron over time with high probability. Their persistence motivates evaluating targeted surgical intervention as a possible way to alleviate plasticity loss.

D.3 KNIFE Performs Better than other Plasticity Injection Methods for MARL Tasks

D.3.1 Sequential One-Step Matrix Game

Figure A7 uses the same deliberately dissimilar task sequence as Figure 1: after learning five payoff matrices with different optimal joint actions, the agents return to the initial task for ten recovery stages. The optimum moves between distant joint actions, while incorrect actions incur

Table A8: Task-wise win rate and neuron ratios on SMACv2 5gen_protoss Task Drift.

Metric	Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
Win Rate (%)	KNIFE	17.26±2.86	31.14±5.66	40.68±1.65	47.93±2.62	51.40±2.24	56.28±2.04	64.48±1.45	70.00±4.23	80.33±3.00	90.18±1.57	94.68±0.74
	KNIFE (Stagnant)	17.33±2.69	22.68±2.62	32.63±3.61	41.22±2.02	48.98±0.83	53.35±1.85	57.95±0.28	68.20±2.14	73.93±1.82	87.70±1.43	91.32±0.78
	QMIX	16.38±1.29	23.62±3.57	34.47±3.64	39.53±1.71	44.27±0.57	48.81±1.77	58.15±2.10	59.07±3.39	68.61±1.31	71.57±2.70	82.75±1.40
Stagnant Ratio (%)	KNIFE	22.22±1.49	24.17±1.49	21.04±2.26	19.65±3.81	20.80±4.12	20.56±7.18	21.18±7.24	23.09±11.06	19.76±6.08	19.17±8.32	22.19±4.05
	KNIFE (Stagnant)	20.33±1.87	26.42±5.77	31.98±2.21	32.22±5.76	34.27±2.93	33.54±0.78	33.75±2.28	39.90±1.49	40.21±3.04	44.48±4.85	47.57±1.90
	QMIX	20.97±3.00	25.38±6.09	36.81±4.96	44.58±4.16	47.64±3.64	46.56±2.51	49.72±6.33	52.22±4.82	55.80±3.36	61.08±3.71	66.53±1.21
Volatile Ratio (%)	KNIFE	12.08±1.87	10.38±0.38	13.47±1.45	15.56±1.75	16.25±2.48	15.94±5.24	13.47±4.55	12.67±6.02	12.15±3.45	10.90±3.07	10.28±2.55
	KNIFE (Stagnant)	15.03±1.28	15.03±1.70	16.15±1.74	15.66±3.12	16.15±1.64	17.50±1.33	17.81±0.95	19.97±3.83	19.55±1.85	20.90±2.46	21.22±1.58
	QMIX	15.38±0.99	16.11±1.75	18.58±1.11	20.00±1.12	19.48±1.74	20.21±1.41	19.86±1.86	20.24±2.55	19.41±2.14	19.24±1.96	18.75±2.21

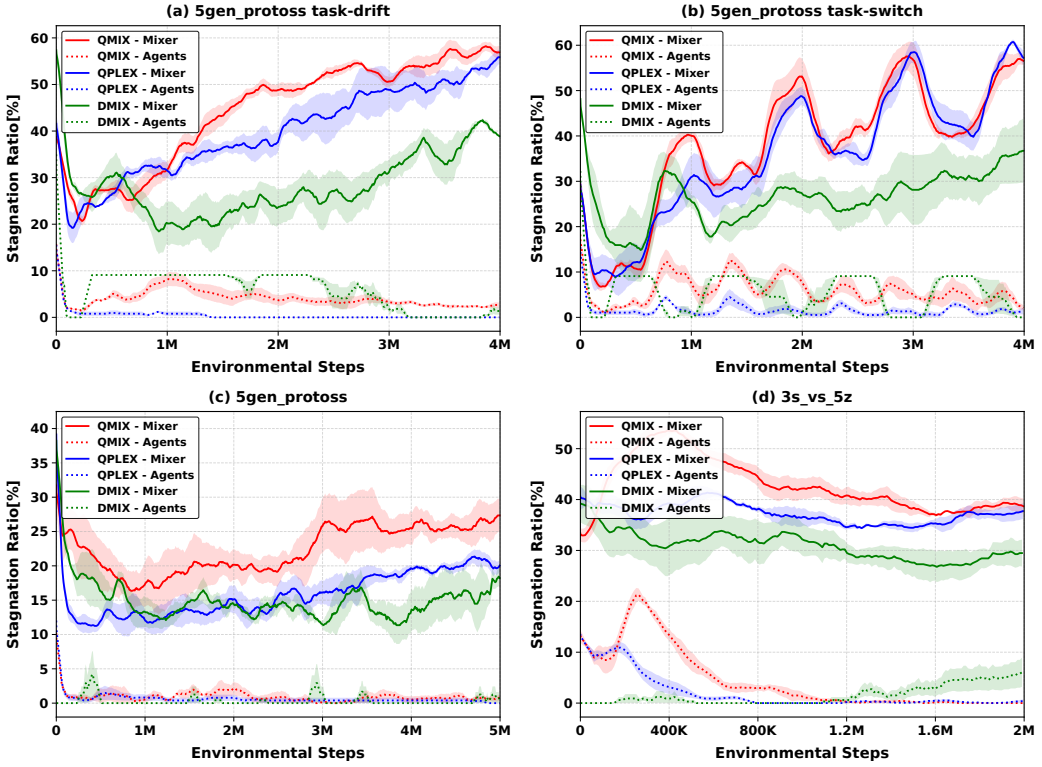

Figure A4: Stagnant-neuron ratios in the mixer and agent networks of QMIX, QPLEX, and DMIX across four settings: (a) SMACv2 5gen_protoss task drift, (b) SMACv2 5gen_protoss task switch, (c) the original non-continual SMACv2 5gen_protoss task, and (d) the SMAC 3s_vs_5z task.

large penalties, leaving little reusable structure across tasks and causing standard QMIX to fail to re-adapt. By comparison, the SMAC and SMACv2 streams in Figure 6 preserve common environment dynamics and coordination strategies such as focus fire. Policies learned on earlier tasks therefore remain partially useful, explaining why the baselines generally degrade rather than collapse in those settings.

We evaluate every intervention in the more severe matrix failure setting. As shown in Figure A8 and the 30-seed results in Table A26, KNIFE progressively reduces the recovery bias to 0.12 ± 0.06 at stage 10. In comparison, QMIX, S&P, ReGraMa, ReDo, CBP, ReSet, and ReBorn retain biases of 14.19, 7.45, 11.58, 14.04, 14.20, 21.09, and 21.55, respectively. Thus, KNIFE remains effective when the underlying learner fails to re-adapt, rather than only in the milder SMAC and SMACv2 streams.

D.3.2 Cyclic One-Step Matrix Game

We further consider a cyclic protocol in which four payoff matrices are repeatedly presented in a fixed order. Unlike the controlled return-to-Task-1 setting in the main text, this protocol repeatedly revisits all tasks and therefore tests whether re-adaptation degrades over successive cycles.

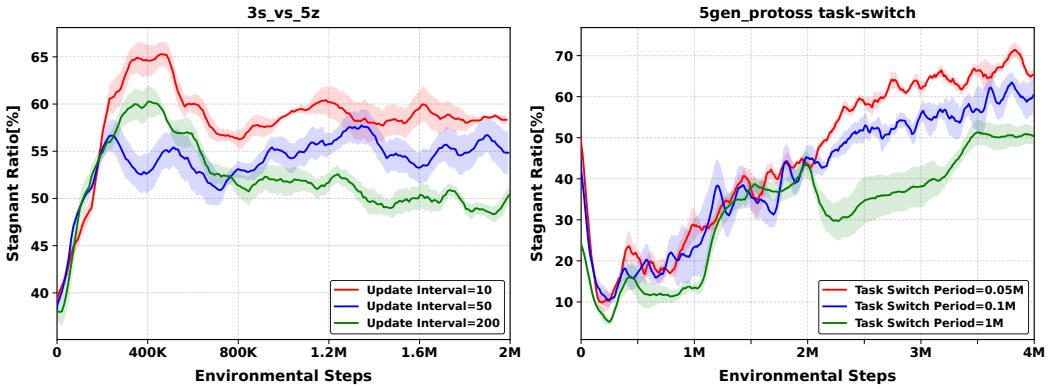

Figure A5: Stagnant ratio under different (left) target network update interval (right) task switching period.

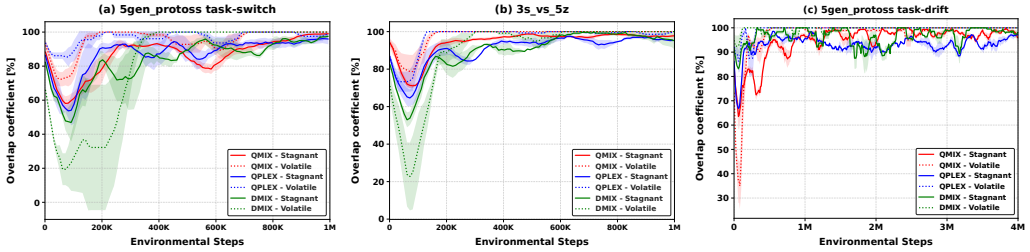

Figure A6: Overlap coefficient stagnant/volatile Neurons between the current iteration and the previous iteration: (a) SMACv2 5 gen_protoss task-switch and (b) SMAC 3s_vs_5z and (right) SMACv2 5 gen_protoss task-drift

In the one-step matrix game with cycling tasks at Figure A9 and Figure A10, KNIFE mitigates plasticity loss and enhances the learning capacity for different tasks. The detailed performance on different tasks is shown in Figure A11.

The detailed data for other benchmarks are shown in Figure A12 and Figure A13.

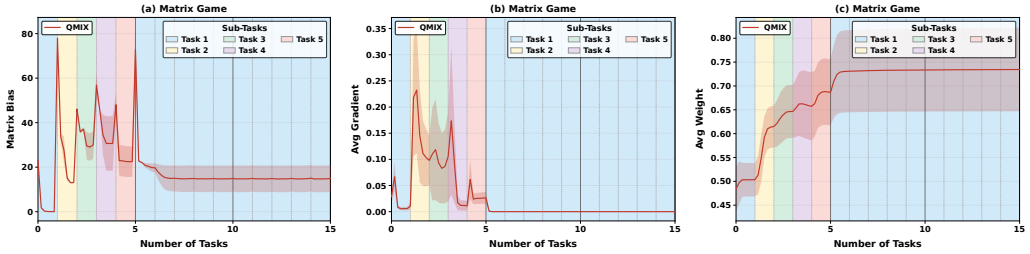

Figure A7: Two agents sequentially learn 5 different tasks, and then they go back to Task 1: (a) the matrix approximation bias (b) the average gradient norm (c) the average weight norm.

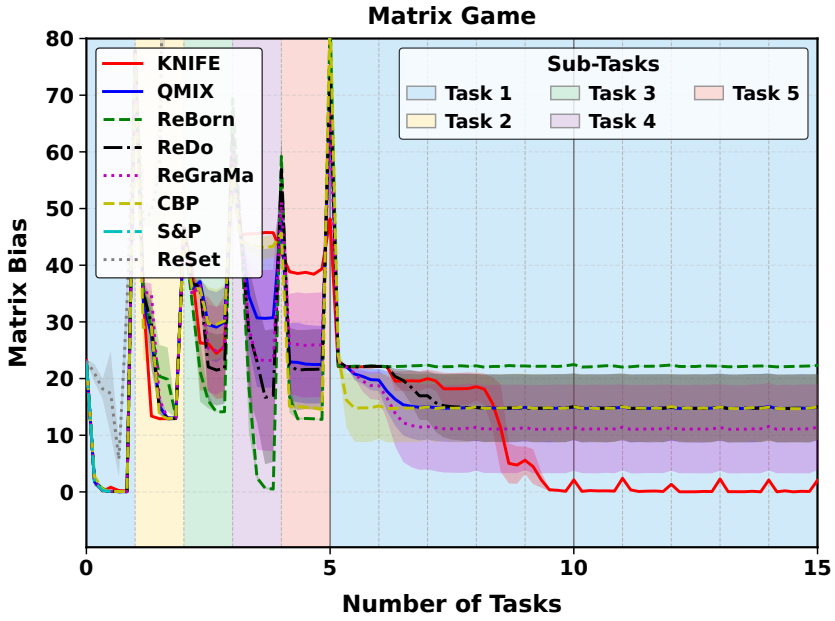

Figure A8: KNIFE performs better than other plasticity injection methods in the one-step matrix game sequential tasks

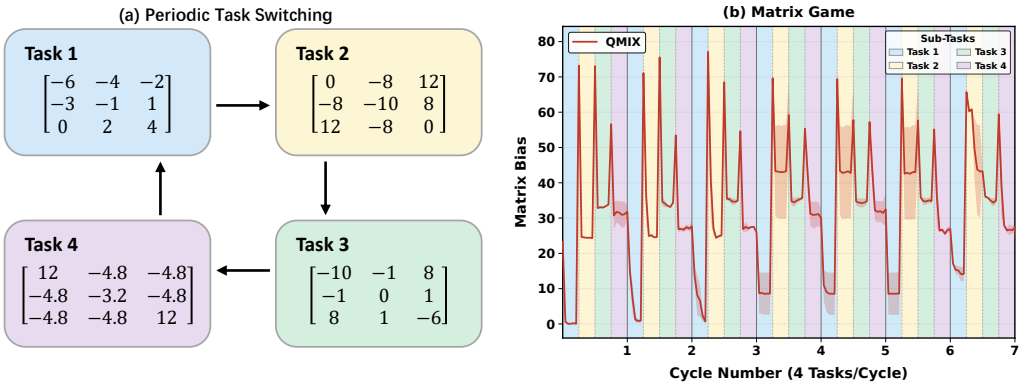

Figure A9: The cyclic setting of the one-step matrix game.

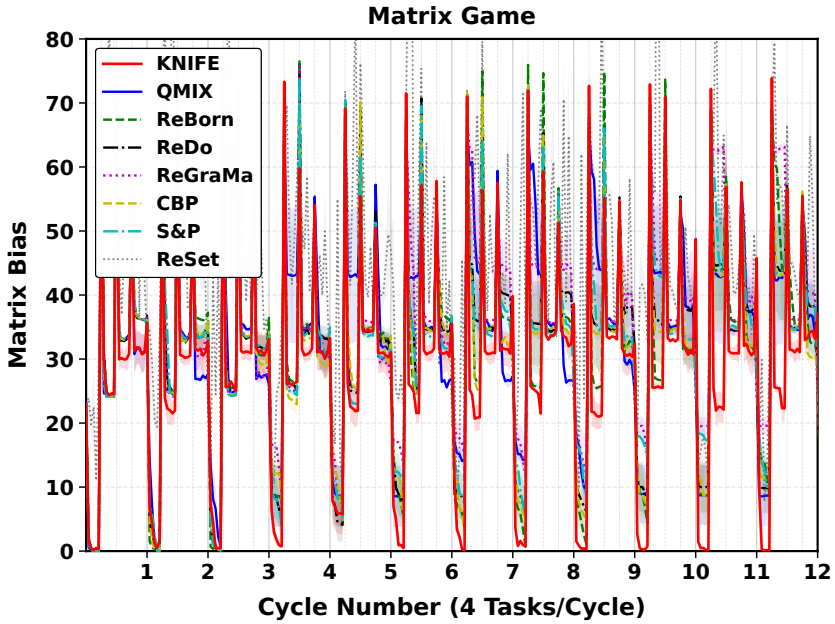

Figure A10: KNIFE outperforms other plasticity injection methods in the cyclic one-step matrix game.

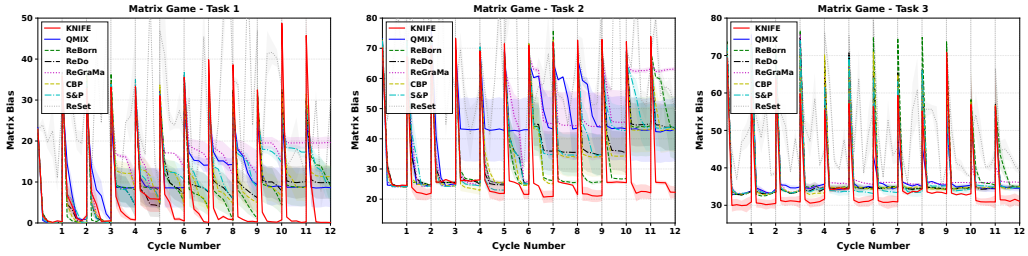

Figure A11: Performance in different tasks of one-step matrix game cyclic setting.

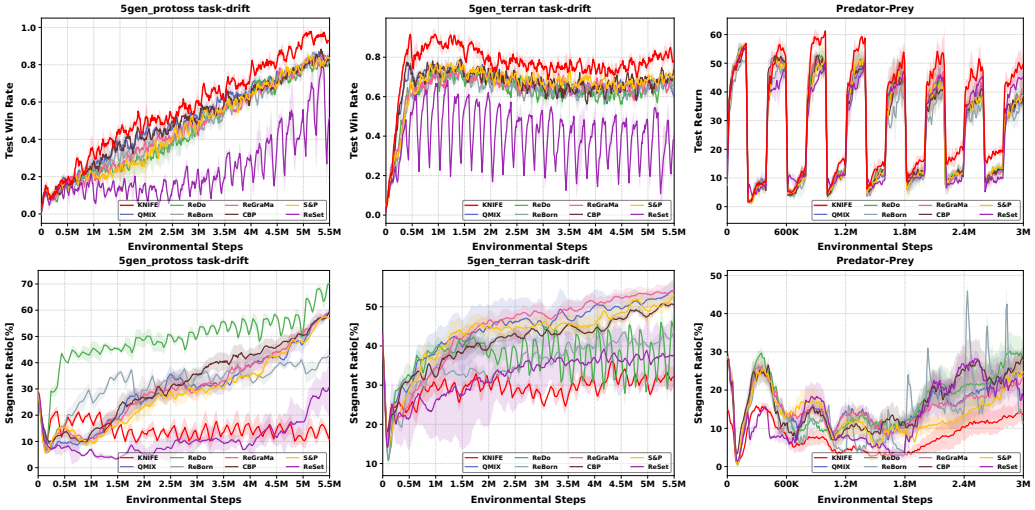

Figure A12: KNIFE performs better than other plasticity injection methods for MARL tasks

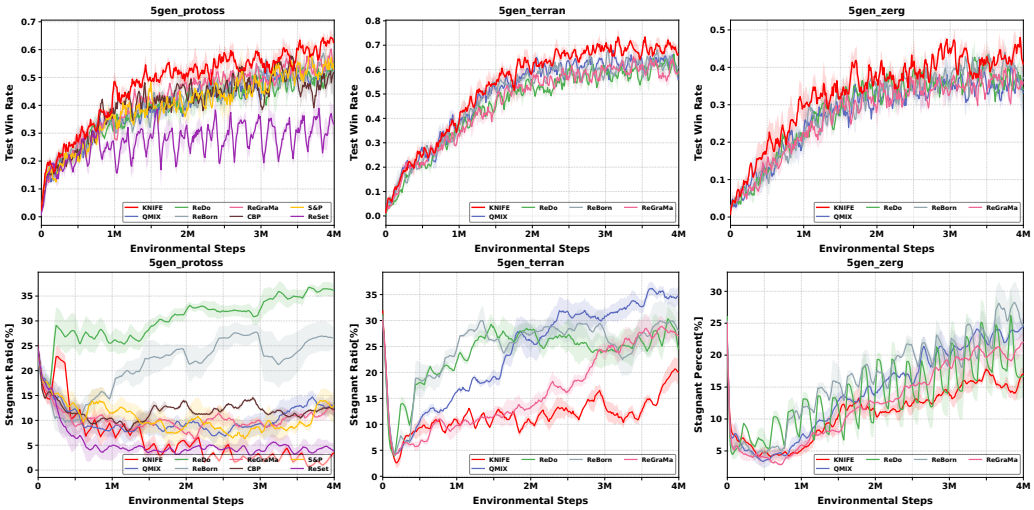

Figure A13: KNIFE performs better than other Neuron-level plasticity injection method in standard SMACv2

Table A9: Task-wise returns on MEAL Level-1 Easy with QMIX (PyTorch, no task head).

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Average
QMIX	1.08 ± 0.40	0.88 ± 0.42	0.79 ± 0.39	0.66 ± 0.18	0.98 ± 0.02	0.53 ± 0.75	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.25 ± 0.36	0.52 ± 0.25
QMIX + ReDo	1.33 ± 0.10	1.07 ± 0.19	1.01 ± 0.16	0.99 ± 0.21	1.05 ± 0.15	1.03 ± 0.11	0.91 ± 0.15	0.55 ± 0.09	0.00 ± 0.00	0.28 ± 0.14	0.82 ± 0.13
QMIX + ReBorn	1.25 ± 0.02	0.99 ± 0.05	0.94 ± 0.05	0.75 ± 0.14	0.94 ± 0.06	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.29 ± 0.21	0.52 ± 0.05
QMIX + PI	1.30 ± 0.08	0.93 ± 0.12	0.98 ± 0.14	0.97 ± 0.14	0.99 ± 0.09	0.94 ± 0.13	0.57 ± 0.04	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.67 ± 0.07
QMIX + KNIFE	$\mathbf{1.36 \pm 0.10}$	$\mathbf{1.35 \pm 0.26}$	$\mathbf{1.33 \pm 0.33}$	$\mathbf{1.06 \pm 0.04}$	$\mathbf{1.25 \pm 0.27}$	$\mathbf{1.25 \pm 0.18}$	$\mathbf{0.94 \pm 0.22}$	$\mathbf{1.09 \pm 0.09}$	$\mathbf{1.22 \pm 0.09}$	$\mathbf{0.92 \pm 0.17}$	$\mathbf{1.18 \pm 0.18}$

Table A10: Task-end normalized performance on MEAL Level-1 Easy (JAX task-head protocol).

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	10-task avg.
QMIX	1.94 ± 0.12	2.28 ± 0.08	1.64 ± 0.03	1.82 ± 0.10	1.52 ± 0.06	1.58 ± 0.05	1.43 ± 0.11	1.37 ± 0.07	1.26 ± 0.09	1.58 ± 0.10	1.64 ± 0.08
ReDo	1.84 ± 0.15	2.34 ± 0.10	1.66 ± 0.12	1.92 ± 0.11	1.28 ± 0.05	1.54 ± 0.04	1.40 ± 0.08	1.37 ± 0.06	1.23 ± 0.03	1.72 ± 0.12	1.63 ± 0.09
ReBorn	1.90 ± 0.11	2.32 ± 0.07	1.64 ± 0.04	1.56 ± 0.12	1.48 ± 0.06	1.04 ± 0.14	1.40 ± 0.07	1.33 ± 0.05	1.30 ± 0.08	1.88 ± 0.10	1.59 ± 0.08
ReGraMa	1.85 ± 0.10	2.39 ± 0.13	1.60 ± 0.03	1.83 ± 0.11	1.52 ± 0.06	1.58 ± 0.05	1.73 ± 0.12	1.37 ± 0.06	1.33 ± 0.05	1.86 ± 0.09	1.71 ± 0.08
S&P	1.90 ± 0.12	2.33 ± 0.10	1.02 ± 0.16	1.92 ± 0.11	1.52 ± 0.06	1.54 ± 0.05	1.33 ± 0.09	1.37 ± 0.06	1.39 ± 0.04	1.98 ± 0.10	1.63 ± 0.09
CBP	1.79 ± 0.09	2.22 ± 0.08	1.60 ± 0.03	1.92 ± 0.10	1.48 ± 0.06	1.55 ± 0.12	1.70 ± 0.11	1.45 ± 0.04	1.30 ± 0.08	1.95 ± 0.11	1.70 ± 0.08
PI	1.90 ± 0.12	2.33 ± 0.10	1.63 ± 0.13	1.79 ± 0.09	1.48 ± 0.06	1.58 ± 0.05	1.73 ± 0.12	1.37 ± 0.06	1.33 ± 0.05	1.98 ± 0.10	1.71 ± 0.19
KNIFE	$\mathbf{2.00 \pm 0.10}$	$\mathbf{2.40 \pm 0.08}$	$\mathbf{1.70 \pm 0.04}$	$\mathbf{2.05 \pm 0.11}$	$\mathbf{1.58 \pm 0.07}$	$\mathbf{1.65 \pm 0.06}$	$\mathbf{1.75 \pm 0.08}$	$\mathbf{1.55 \pm 0.04}$	$\mathbf{1.42 \pm 0.06}$	$\mathbf{2.40 \pm 0.12}$	$\mathbf{1.85 \pm 0.08}$

D.4 Evaluation on the MEAL Benchmark

D.4.1 Continual MARL Benchmark ($n = 2$)

Tables A9–A13 report five complementary MEAL evaluations with distinct implementations and purposes. Table A9 is the only evaluation implemented in PyTorch: it uses Level-1 Easy ($n = 2$), a QMIX backbone, and no task head. Tables A10 and A11 use the original JAX implementation with its task head on Level-1 Easy and Medium, respectively. Table A12 also uses the original MEAL JAX implementation and its default MAPPO backbone for an intervention-target ablation, whereas Table A13 evaluates the single-agent setting ($n = 1$) with PPO. Because the tables differ in backend, backbone, environment difficulty, task-head design, or agent count, each supports comparisons among methods within that table; absolute values should not be compared directly across distinct protocols. For the JAX results in Tables A10, A11, and A12, we follow the standard MEAL evaluation protocol, including its task-specific head architecture. Only the complementary PyTorch experiment in Table A9 deliberately uses a shared QMIX network without a task head. KNIFE itself does not rely on task-boundary signals for intervention; its diagnosis rule and global periodic intervention schedule remain unchanged across both protocols. The $n = 2$ configuration is a continual MARL setting and is used to test KNIFE on a standard multi-agent task sequence rather than only on our modified benchmarks. KNIFE retains the same intervention hyperparameters throughout, without MEAL-specific or task-specific tuning. Its interval is fixed at $T = 0.2$M environment steps, so the first intervention occurs within Task 1 rather than after the first task. The complementary PyTorch experiment uses 10 random seeds; its entries report means and 95% confidence intervals.

As shown in Table A9, QMIX equipped with KNIFE achieves the highest average return of 1.18, compared with 0.52 for QMIX alone. While QMIX, QMIX + ReBorn, and QMIX + PI exhibit severe degradation on later tasks, QMIX + KNIFE maintains non-zero performance throughout the sequence. This result shows that KNIFE remains effective on an established continual MARL benchmark, rather than only on our modified SMACv2 task streams. In particular, QMIX reaches zero return on Tasks 7–9, QMIX + ReBorn on Tasks 6–9, and QMIX + PI on Tasks 8–10, whereas QMIX + KNIFE remains competent on every task. Together with the return-to-initial-task matrix experiment in Figure A8, this shows that KNIFE remains useful when the underlying MARL learner suffers a catastrophic adaptation failure. The higher Task-1 return should not be interpreted as a pre-intervention advantage: Task 1 is already an intervention stage for KNIFE, whose first surgery occurs after the fixed interval $T = 0.2$M environment steps. The gain is consistent with our single-task results, which show that stagnant neurons can arise and be repaired even before any task transition.

Within the task-head Level-1 Easy setting, KNIFE achieves the highest 10-task average of 1.85, compared with 1.64 for QMIX and 1.71 for PI and ReGraMa, the strongest competing plasticity baselines. Thus, KNIFE attains the best average, while several baselines remain competitive in this easier setting.

Using the same task-head evaluation protocol, we next evaluate the harder MEAL Level-1 Medium setting ($n = 2$). This remains distinct from the PyTorch no-task-head experiment in Table A9.

Within this JAX Level-1 Medium setting, KNIFE obtains the highest 10-task average of 1.70, compared with 1.39 for QMIX and 1.52 for PI. The improvement is more pronounced than on

Table A11: Task-end normalized performance on MEAL Level-1 Medium (JAX task-head protocol).

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Average
QMIX	2.08 ± 0.12	1.45 ± 0.06	1.36 ± 0.14	1.39 ± 0.09	1.69 ± 0.25	1.43 ± 0.41	0.91 ± 0.33	0.58 ± 0.76	1.53 ± 0.11	1.51 ± 0.78	1.39 ± 0.10
ReDo	1.84 ± 0.42	1.48 ± 0.07	1.09 ± 0.28	1.33 ± 0.05	1.61 ± 0.66	1.77 ± 0.50	1.44 ± 0.08	1.36 ± 0.15	1.48 ± 0.11	1.49 ± 0.29	1.49 ± 0.04
ReBorn	2.05 ± 0.13	1.49 ± 0.06	1.36 ± 0.14	1.35 ± 0.12	1.65 ± 0.17	1.79 ± 0.41	0.92 ± 0.34	1.08 ± 0.11	1.36 ± 0.33	1.40 ± 0.10	1.45 ± 0.12
ReGraMa	1.80 ± 0.43	1.45 ± 0.06	$\mathbf{1.47 \pm 0.02}$	1.39 ± 0.09	1.73 ± 0.36	1.72 ± 0.11	1.35 ± 0.08	1.23 ± 0.14	1.36 ± 0.20	1.48 ± 0.18	1.50 ± 0.13
S&P	2.05 ± 0.12	1.43 ± 0.07	1.46 ± 0.02	1.33 ± 0.15	1.15 ± 0.27	1.29 ± 0.12	1.34 ± 0.27	1.19 ± 0.13	1.39 ± 0.14	1.63 ± 1.12	1.43 ± 0.05
CBP	2.03 ± 0.17	1.45 ± 0.06	1.42 ± 0.03	1.46 ± 0.03	1.69 ± 0.07	0.83 ± 0.41	1.01 ± 0.30	$\mathbf{1.39 \pm 0.21}$	1.67 ± 0.60	1.45 ± 0.77	1.44 ± 0.12
PI	1.99 ± 0.21	1.39 ± 0.07	1.16 ± 0.02	1.42 ± 0.03	1.58 ± 0.18	1.79 ± 0.12	1.41 ± 0.17	1.32 ± 0.15	1.42 ± 0.14	1.75 ± 0.25	1.52 ± 0.01
KNIFE	$\mathbf{2.31 \pm 0.19}$	$\mathbf{1.50 \pm 0.06}$	$\mathbf{1.47 \pm 0.02}$	$\mathbf{1.49 \pm 0.13}$	$\mathbf{1.80 \pm 0.47}$	$\mathbf{1.84 \pm 0.42}$	$\mathbf{1.45 \pm 0.27}$	$\mathbf{1.39 \pm 0.22}$	$\mathbf{1.95 \pm 0.18}$	$\mathbf{1.76 \pm 0.23}$	$\mathbf{1.70 \pm 0.09}$

Table A12: Task-wise returns for the MEAL Level-1 MAPPO intervention-target ablation.

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Average
MAPPO	1.6217±0.2716	1.7073±0.3076	1.7368±0.3405	1.7893±0.3796	1.7274±0.3755	1.8970±0.4134	1.7178±0.4634	1.9130±0.4379	1.9310±0.3509	2.0551±0.2019	1.8096±0.3542
MAPPO + KNIFE (STAGNANT)	1.9826±0.3729	1.9800±0.2409	1.9815±0.4342	**2.2044±0.1843**	**2.0816±0.1472**	1.9782±0.5412	**2.0843±0.0848**	1.8475±0.4629	1.9940±0.4264	2.0991±0.4043	2.0233±0.3299
MAPPO + KNIFE (Full)	**1.9952±0.4061**	**1.9972±0.3253**	**2.2721±0.1926**	2.1862±0.1195	2.0722±0.2222	**2.3281±0.1650**	2.0764±0.3243	**1.9877±0.4822**	**2.2670±0.3273**	**2.2374±0.1403**	**2.1410±0.2705**

Easy, indicating that KNIFE's benefit becomes clearer as the continual adaptation problem becomes harder. The backend and task-head difference still preclude direct numerical comparison with the no-task-head results in Table A9.

D.4.2 Intervention-Target Ablation with MAPPO on MEAL

To examine whether stagnant-neuron intervention alone is sufficient under natural training dynamics, we use the default MEAL MAPPO backbone for all variants within the original MEAL JAX implementation and retain its task-specific head. This shares the standard MEAL evaluation protocol used in Tables A10 and A11, while using a different backbone and experimental purpose. It is distinct from the PyTorch no-task-head setting in Table A9. We compare MAPPO, MAPPO + KNIFE (Stagnant), which repairs only naturally detected stagnant neurons, and MAPPO + KNIFE (Full), which repairs both stagnant and volatile neurons.

As shown in Table A12, on the MAPPO backbone, adding stagnant-neuron repair improves the average return from 1.8096 to 2.0233, while full KNIFE further improves it to 2.1410. The full intervention achieves a higher mean return than MAPPO + KNIFE (Stagnant) on seven of the ten tasks. These results suggest that stagnant-neuron repair accounts for a substantial portion of the improvement in this ablation, while volatile-neuron intervention provides an additional complementary benefit.

D.4.3 Extension to Single-Agent PPO

We further evaluate KNIFE on MEAL Level-1 ($n = 1$) with PPO, using ten sequential tasks, 100M environment steps per task, and 10 random seeds. Unlike the $n = 2$ configuration, this is a single-agent RL setting. Its purpose is to test whether KNIFE's benefit transfers beyond MARL, not to claim that unmodified PPO catastrophically fails on every task. PPO remains competent throughout this sequence, so improvements here provide evidence of continual adaptation benefits on this benchmark beyond merely recovering from complete performance collapse.

As shown in Table A13, KNIFE improves the average return from 1.19 to 1.37, while ReDo obtains 1.04. This improvement is accompanied by a substantial reduction in stagnant neurons: Table A14 shows that the average stagnant-neuron ratio decreases from 46.12% with PPO to 30.15% with KNIFE, whereas ReDo increases it to 56.22%. This provides an explanation for PPO outperforming ReDo: ReDo's dormant-neuron reset does not reduce the pathology measured by RUA on this backbone and instead coincides with a larger stagnant-neuron fraction. By contrast, KNIFE directly targets these neurons. These results suggest that the stagnant-neuron phenomenon and the benefit of targeted intervention are not restricted to MARL. KNIFE is not best on every individual task—PPO is slightly better on Task 9—but it improves nine of ten task returns and the sequence average, supporting its usefulness in this single-agent benchmark rather than establishing general effectiveness across continual RL. As in the MARL experiments, PPO Task 1 is not intervention-free for KNIFE: the intervention clock begins at the start of Task 1 and first triggers after the fixed interval $T = 0.2$M environment steps. Its Task-1 result therefore measures the effect of early single-task stagnant-neuron repair rather than an initialization advantage.

Table A13: Task-wise returns on single-agent MEAL Level-1 with PPO.

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Average
PPO	1.20±0.07	1.17±0.07	1.13±0.13	1.11±0.11	1.27±0.25	1.22±0.32	1.13±0.13	1.19±0.31	**1.28±0.17**	1.22±0.24	1.19±0.18
ReDo	1.08±0.04	1.06±0.09	1.15±0.03	1.17±0.09	0.76±0.52	1.12±0.01	1.11±0.01	0.80±0.46	1.21±0.26	0.99±0.19	1.04±0.17
KNIFE	**1.29±0.29**	**1.61±0.16**	**1.48±0.23**	**1.46±0.26**	**1.30±0.18**	**1.40±0.20**	**1.42±0.15**	**1.21±0.26**	1.25±0.32	**1.30±0.42**	**1.37±0.25**

Table A14: Task-wise stagnant-neuron ratios on single-agent MEAL Level-1 with PPO.

Method	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Average
PPO	30.24 ± 2.10	33.62 ± 5.73	49.26 ± 4.65	40.14 ± 3.01	45.98 ± 15.32	50.90 ± 6.16	47.79 ± 10.93	61.98 ± 4.93	45.97 ± 1.54	55.36 ± 4.65	46.12 ± 5.90
ReDo	34.52 ± 0.94	42.79 ± 3.47	49.80 ± 0.69	51.51 ± 10.75	59.33 ± 6.06	65.57 ± 3.56	61.97 ± 9.07	67.85 ± 0.16	61.06 ± 2.50	67.82 ± 5.14	56.22 ± 4.23
KNIFE	**25.59 ± 1.67**	**27.05 ± 7.32**	**35.46 ± 4.79**	**27.99 ± 8.56**	**28.12 ± 2.50**	**33.64 ± 5.77**	**31.73 ± 6.63**	**30.35 ± 1.92**	**35.97 ± 4.42**	**25.58 ± 9.60**	**30.15 ± 5.32**

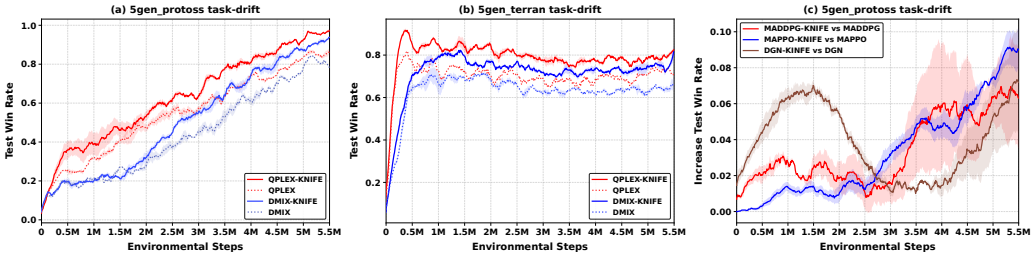

Figure A14: Performance improvements from KNIFE across MARL learners. The left and middle panels compare QPLEX and DMIX with their KNIFE variants on the SMACv2 5gen_protoss and 5gen_terran task-drift settings, respectively. The right panel reports the win-rate improvements from adding KNIFE to MADDPG, MAPPO, and DGN on SMACv2 5gen_protoss task drift.

D.5 Extension of KNIFE to Additional MARL Learners

The main text reports a compact cross-learner comparison in Table 2. Here we provide full uncertainty estimates and additional curves. Stagnant neurons are observed during sequential task transfers beyond hypernetwork-generated mixing weights, motivating evaluation on additional MARL learners.

To test whether KNIFE can be transferred beyond value-decomposition learners, we integrate it into a multi-agent proximal policy optimization method (MAPPO), a multi-agent deep deterministic policy gradient method (MADDPG), and a communication-centric actor-critic method (DGN). For all three learners, RUA detection and KNIFE are applied to the hidden fully connected layers of both actor and critic networks, excluding output heads. We use the same configuration, $\alpha = 0.25$, $\beta = 3$, $\gamma = 0.995$, and $T = 0.2$M environment steps, without learner-specific tuning.

Table A15 reports means and 95% confidence intervals across these learners under the same sequential task-transfer setting. Equipping each learner with KNIFE improves performance in this evaluation, and KNIFE also outperforms the compared neuron-level reset interventions, ReDo and ReBorn. These results demonstrate that KNIFE can be integrated with multiple MARL learner families beyond value factorization. Broader evaluation across additional task streams and architectures is left for future work.

Table A15: Win rate (%) across MARL learner families on 5gen_protoss task drift.

Baseline	Baseline Performance	+ ReDo	+ ReBorn	+ KNIFE (Ours)
MAPPO	18.84 ± 2.59	20.26 ± 4.25	17.61 ± 0.91	$\mathbf{25.44 \pm 2.69}$
MADDPG	20.13 ± 2.66	10.48 ± 5.26	29.16 ± 2.27	$\mathbf{34.55 \pm 3.10}$
DGN	17.48 ± 1.66	20.58 ± 2.32	19.85 ± 2.10	$\mathbf{22.73 \pm 1.04}$

D.6 De-hypernetworked Ablation: Structural vs. Training Dynamics

In value factorization methods like QMIX, mixing weights are typically generated by hypernetworks. To investigate whether stagnant neurons are specific to this architectural choice or also occur with alternative mixing architectures, we conducted a "de-hypernetworked" ablation.

Specifically, in the *5gen_protoss task-switch* environment, we replaced the hypernetwork with alternative architectures: a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN). To characterize the accompanying training dynamics, we tracked not only the stagnation ratio but also the Weight Spectra Entropy (measuring the diversity and rank of the weight matrices) and the Gradient Norm (reflecting gradient activity) across training.

Table A16: Stagnant ratio, weight-spectrum entropy, and gradient norm across mixer architectures.

Metric	Architecture	0.5M step	1M step	1.5M step	2M step	2.5M step	3M step
Stagnant Ratio (%)	**Hypernetwork (Default)**	13.2	31.4	27.1	32.8	35.1	44.9
	MLP	26.6	31.4	29.8	34.0	30.7	34.6
	CNN	53.3	62.1	61.3	61.5	58.5	59.7
Weight Spectra Entropy	**Hypernetwork (Default)**	4.006	3.968	3.953	3.937	3.938	3.933
	MLP	3.941	3.911	3.882	3.871	3.859	3.850
	CNN	3.893	3.862	3.846	3.829	3.823	3.813
Gradient Norm	**Hypernetwork (Default)**	0.0291	0.0353	0.0307	0.0335	0.0242	0.0285
	MLP	0.0418	0.0310	0.0264	0.0227	0.0197	0.0207
	CNN	0.0157	0.0215	0.0132	0.0141	0.0102	0.0107

As shown in Table A16, stagnant neurons persist across all architectures. Notably, replacing the hypernetwork with a CNN or MLP does not eliminate the pattern; in fact, the CNN structure exhibits an even higher stagnation ratio. Furthermore, across all architectures, the Weight Spectra Entropy steadily decreases (indicating a loss of feature diversity and representational expressivity), while the Gradient Norm diminishes progressively over time. These co-occurring trends support the interpretation that stagnation is not merely a structural artifact unique to hypernetworks, although they do not establish it as the unique cause of continual-MARL degradation.

D.7 Initialization Strategies: Methodological Comparison and Ablation

The effectiveness of plasticity injection depends on how newly introduced parameters are integrated with the existing frozen network. KNIFE and PI use related function-preserving-at-injection constructions but differ in the granularity and allocation of plastic capacity: PI operates at the layer level, whereas KNIFE allocates active capacity locally to detected neurons and conditions its initialization on the surrounding network.

PI: State-Agnostic Kaiming Initialization. PI typically employs standard Kaiming initialization for both weights W and biases b of the injected layers. The initialization follows:

$$W \sim \mathcal{N}\left(0, \frac{2}{d_{in}}\right), \quad b \sim \mathcal{U}\left(-\frac{6}{\sqrt{d_{in}}}, \frac{6}{\sqrt{d_{in}}}\right) \tag{D.20}$$

where d_{in} is the input dimension. This approach is state-agnostic because it does not condition on the current optimization state of the pre-existing layers. In some settings, newly injected neurons may consequently enter activation dead zones (e.g., dying ReLUs) or initially propagate weak signals during transfer learning.

KNIFE: State-Aware Initialization. In contrast, KNIFE applies a state-aware strategy designed to promote usable activations and gradient flow for newly injected branches:

- **Incoming Weights:** The active branch uses Kaiming initialization, $\mathbf{w}_{in,i}^{\ell,\text{active}} \sim \mathcal{N}(0, \delta_{in}^2)$ with $\delta_{in} = \sqrt{2/d_{\ell-1}}$.
- **Positive Input Bias:** We initialize $b_i^{\ell,\text{active}} = 0.5\delta_{in}$. This choice is designed to increase the likelihood of positive activations and facilitate early gradient updates.
- **Contextual Output Weight:** The outgoing weights are initialized as $\mathbf{w}_{out,i}^{\ell,\text{active}} \sim \mathcal{N}(0, \delta_{out}^2)$, where $\delta_{out} = \max(\text{std}(w^{\ell+1}), 10^{-2})$. This choice is intended to align the initial forward-signal scale with the existing network distribution while avoiding a vanishingly small scale.

Ablation Study. To evaluate the performance contribution of this state-aware design, we consider an ablated version, *KNIFE (Kaiming init)*, which uses standard Kaiming initialization instead of our

proposed approach. We compare its performance against the full KNIFE algorithm, PI, and ReDo across different sequential task transfer environments.

Table A17: Win rate (%) for initialization-strategy ablations.

Environment	KNIFE (Ours)	KNIFE (Kaiming init)	PI @ 25%	PI @ 50%	ReDo
5gen_protoss task-switch	$\mathbf{91.17 \pm 1.17}$	88.43 ± 0.97	82.55 ± 0.33	84.45 ± 1.38	81.20 ± 1.36
5gen_protoss task-drift	$\mathbf{94.68 \pm 0.74}$	89.49 ± 1.44	81.48 ± 0.87	82.27 ± 1.19	79.19 ± 0.63
5gen_protoss	$\mathbf{60.89 \pm 1.87}$	57.91 ± 0.04	53.41 ± 1.07	54.02 ± 2.85	50.33 ± 2.49

As shown in Table A17, KNIFE achieves the highest overall performance and consistently outperforms the KNIFE (Kaiming init) baseline. This demonstrates a performance benefit from the proposed initialization and is consistent with its intended role. However, this ablation does not directly establish effects on activation ratios, gradient flow, update magnitudes, or dead-neuron fractions.

Furthermore, even when reverting to standard Kaiming initialization, KNIFE consistently outperforms PI across the tested environments. This result suggests that targeted neuron-level surgery provides benefits beyond the initialization strategy in these settings.

D.8 Ablations on Neuron-Selection Criteria and Repair Mechanisms

We disentangle two components of the intervention: neuron selection and the repair mechanism. For target-selection comparisons, we hold the KNIFE repair mechanism fixed while changing the selected neurons; RUA and GraMa use matched repair budgets. For repair-mechanism comparisons, KNIFE, ReDo, and ReBorn operate on exactly the same RUA-identified stagnant neurons. We evaluate the following variants:

- **KNIFE (Stagnant):** Applies KNIFE surgery only to stagnant neurons, whereas full KNIFE targets both stagnant and volatile neurons.
- **KNIFE (non-stagnant):** Applies KNIFE surgery to non-stagnant neurons, keeping stagnation constant.
- **KNIFE (Dormant):** Applies KNIFE specifically to dormant neurons (inactive units).
- **KNIFE (GraMa):** Uses the GraMa criterion to select intervention targets while retaining the KNIFE repair mechanism.
- **ReDo (Stagnant) & ReDo (Dormant):** Applies the standard ReDo reset mechanism to stagnant and dormant neurons, respectively.
- **ReBorn (Stagnant) & ReBorn (Dormant):** Applies the ReBorn repair mechanism to stagnant and dormant neurons, respectively.

As demonstrated in Table A18, KNIFE (Stag.) outperforms KNIFE (GraMa), KNIFE (Dorm.), and KNIFE (Non-stag.) under a controlled repair mechanism, indicating that RUA-identified stagnant neurons are the most effective target among the evaluated selection criteria. More importantly, the result is not merely attributable to the RUA detector: with exactly the same RUA-identified stagnant neurons as intervention targets, KNIFE consistently outperforms both ReDo and ReBorn across all three settings. This comparison isolates the benefit of KNIFE's neuron-local, function-preserving-at-injection repair operator from its detector. These results provide intervention-based evidence that stagnant neurons are a useful and actionable repair target associated with plasticity loss; they do not establish stagnant neurons as its unique or primary cause.

D.9 Network Capacity Control

KNIFE temporarily introduces additional neurons during training, raising the question of whether its improvement simply results from increased network capacity. To control for this factor, we enlarge the initial mixing-network width of QMIX and ReBorn to $1.2\times$ and $1.5\times$ their default sizes and train them under the same continual settings. We also report PI, which introduces additional parameters through layer-level plasticity injection.

Table A18: Win rate (%) for controlled target-selection and repair-mechanism ablations.

Environment	KNIFE	KNIFE (Stag.)	KNIFE (GraMa)	KNIFE (Dorm.)	KNIFE (Non-stag.)	ReDo (Stag.)	ReDo (Dorm.)	ReBorn (Stag.)	ReBorn (Dorm.)
5gen_protoss	$\mathbf{60.89 \pm 1.87}$	58.60 ± 1.60	56.21 ± 1.72	55.37 ± 3.09	46.09 ± 2.10	55.20 ± 2.30	50.33 ± 2.49	53.41 ± 4.94	51.46 ± 5.05
5gen_protoss Task Switch	$\mathbf{91.17 \pm 1.17}$	89.70 ± 1.10	87.13 ± 1.33	87.05 ± 2.30	77.46 ± 0.41	88.01 ± 0.63	81.20 ± 1.36	83.67 ± 0.46	79.65 ± 1.34
5gen_protoss Task Drift	$\mathbf{94.68 \pm 0.74}$	91.32 ± 0.78	88.46 ± 1.27	85.41 ± 0.64	76.00 ± 1.30	86.71 ± 1.63	79.19 ± 0.63	86.18 ± 0.95	79.74 ± 2.53

Table A19: Win rate (%) for network-capacity controls on SMACv2 5gen_protoss.

Environment	KNIFE	PI	QMIX	QMIX $1.2\times$	QMIX $1.5\times$	ReBorn	ReBorn $1.2\times$	ReBorn $1.5\times$
Task Drift	$\mathbf{94.68 \pm 0.74}$	82.27 ± 1.19	82.75 ± 1.40	81.77 ± 1.41	83.12 ± 1.38	84.17 ± 2.23	83.75 ± 0.74	83.40 ± 2.71
Task Switch	$\mathbf{91.17 \pm 1.17}$	84.45 ± 1.38	82.50 ± 2.50	85.67 ± 3.21	84.79 ± 4.85	79.65 ± 1.34	81.89 ± 4.46	83.49 ± 2.17

As shown in Table A19, simply increasing the initial mixing-network width does not reproduce the gains of KNIFE. Enlarging QMIX or ReBorn to $1.2\times$ or $1.5\times$ yields only limited and inconsistent changes, while KNIFE remains substantially better in both Task Drift and Task Switch. This indicates that KNIFE's improvement cannot be explained by increased network capacity alone, but depends on its targeted plasticity intervention.

D.10 Computational Overhead and Memory Efficiency

To address potential concerns regarding the computational overhead and memory pressure introduced by our composite units, we comprehensively profiled the peak GPU memory consumption (in MB) and total training time (in hours) of KNIFE against both layer-level (PI) and neuron-level (ReDo, ReBorn) baselines.

KNIFE does not pre-allocate three additional layers. A layer starts with the same width and parameter count as its baseline counterpart. If its initial width is d_ℓ and k_ℓ neurons are repaired, the original selected neurons serve as frozen knowledge branches, while one active and one compensation neuron are dynamically appended or assigned to reusable slots for each selection. The temporary width is therefore at most $d_\ell + 2k_\ell$, rather than $4d_\ell$; even repairing every neuron at once yields width $3d_\ell$. Before the first intervention, KNIFE has no parameter overhead. After injection, the overhead is proportional to the number of repaired neurons; after every learner update, the frozen knowledge and compensation parameters are decayed and branches below the pruning threshold are removed and recycled.

As shown in Table A20, PI's layer-level expansion nearly doubles the memory footprint (e.g., reaching over 630 MB in the task-switch setting). Conversely, KNIFE's targeted neuron-level surgery introduces only a minimal memory increase compared to the standard QMIX baseline. Furthermore, when compared with other neuron-level methods (Table A21), although KNIFE requires a moderate temporary memory expansion during the injection phase, its continuous decay and tensor reuse mechanisms keep overall utilization highly efficient. Crucially, KNIFE achieves the highest win rates and the shortest total training times. These results indicate that KNIFE improves adaptability with moderate computational and memory overhead.

Table A20: Win rate and peak memory for layer-level injection baselines.

Environment	Metric	KNIFE (Ours)	PI @ 25%	PI @ 50%	QMIX (Baseline)
5gen_protoss task-switch	Win Rate (%)	91.17 ± 1.17	82.55 ± 0.33	84.45 ± 1.38	82.50 ± 2.50
	Memory (MB)	330	654	630	293
5gen_protoss task-drift	Win Rate (%)	94.68 ± 0.74	81.48 ± 0.87	82.27 ± 1.19	82.75 ± 1.40
	Memory (MB)	431	655	679	416
5gen_protoss	Win Rate (%)	60.89 ± 1.87	53.41 ± 1.07	54.02 ± 2.85	50.98 ± 3.55
	Memory (MB)	176	239	230	166

D.11 Scalability to Large-Scale

To explicitly evaluate whether KNIFE can effectively transfer to larger-scale MARL settings we designed an extended sequential continual setting:

Table A21: Win rate, training time, and peak memory for neuron-level methods.

Environment	Metric	KNIFE (Ours)	ReDo	ReBorn
5gen_protoss	Win Rate (%)	$\mathbf{60.89 \pm 1.87}$	50.50 ± 2.05	53.41 ± 2.94
	Time (h)	$\mathbf{26.01}$	27.70	30.73
	Memory (MB)	176.63	166.88	164.45
5gen_protoss task-switch	Win Rate (%)	$\mathbf{91.17 \pm 1.17}$	81.20 ± 1.36	79.65 ± 1.34
	Time (h)	$\mathbf{25.98}$	29.34	31.10
	Memory (MB)	330.58	315.36	300.93

- **Large-Scale Agent Scaling:** The environment progressively scales up both the number of agents and enemies from 5 to 40 ($5m \rightarrow 10m \rightarrow 20m \rightarrow 30m \rightarrow 40m$).

As shown in Table A22, scaling the environment up to 40 agents significantly challenges the plasticity of the baseline methods.

Table A22: Win rate (%) in the 5m-to-40m scaling sequence.

Sequential Setting	QMIX	ReDo	KNIFE (Ours)
5m → 10m → 20m → 30m → 40m	66.45 ± 17.55	39.06 ± 13.43	$\mathbf{83.59 \pm 5.52}$

In the evaluated sequence, KNIFE achieves a win rate of 83.59% and outperforms the compared QMIX and ReDo baselines. This result provides evidence that KNIFE can remain effective as the setting scales to 40 agents, while broader large-scale generalization requires evaluation on additional task sequences and learners.

D.12 Parameter Sensitivity

We vary one KNIFE hyperparameter at a time on MEAL Level-1 Easy while keeping the remaining hyperparameters fixed at the default configuration ($\alpha = 0.25$, $\beta = 3$, $\gamma = 0.995$, and $T = 0.2$M). Results are 10-task averages reported as means and 95% confidence intervals over five seeds. The default result is the same as that reported for KNIFE in Table A10. Table A23 summarizes the resulting 10-task averages.

Performance varies by at most 0.041 from the default across the tested configurations, and several alternatives slightly improve the average. These results indicate that KNIFE is stable across the evaluated ranges of α, β, γ, and T without requiring precise tuning.

To examine whether the comparison depends on the published default hyperparameters of the competing neuron-level methods, we additionally conduct equal-budget one-dimensional sweeps over their primary neuron-selection thresholds. The default results are aligned with Table A10, and all methods use the same evaluation protocol and tuning budget. Results are means and 95% confidence intervals over five seeds.

The tuned variants change the corresponding averages by at most 0.019 and remain below KNIFE's default average of 1.850. Thus, KNIFE's advantage in this setting is not explained by comparing against a single unfavorable threshold choice for these baselines.

D.13 Statistical Robustness Across Random Seeds

To examine whether our empirical conclusions are sensitive to random initialization, we conduct two complementary higher-seed evaluations. First, on the SMACv2 5gen_protoss Task Drift setting, we evaluate all methods using 5, 10, and 20 independent random seeds. As shown in Table A25, increasing the number of seeds changes the absolute means and confidence intervals, but does not alter the overall ranking: KNIFE achieves the highest win rate under all three evaluation budgets.

Table A23: KNIFE hyperparameter sensitivity on MEAL Level-1 Easy (10-task average).

Hyperparameter	Configuration	10-task Avg.	Δ vs. Default
α	0.10	1.841 ± 0.113	-0.009
	0.25 (default)	$\mathbf{1.850 \pm 0.080}$	0.000
	0.40	1.809 ± 0.018	-0.041
β	2	1.831 ± 0.085	-0.019
	3 (default)	$\mathbf{1.850 \pm 0.080}$	0.000
	5	1.855 ± 0.080	0.005
γ	0.990	1.838 ± 0.079	-0.012
	0.995 (default)	$\mathbf{1.850 \pm 0.080}$	0.000
	1.000	1.837 ± 0.080	-0.013
Intervention interval T	0.1M	1.845 ± 0.142	-0.005
	0.2M (default)	$\mathbf{1.850 \pm 0.080}$	0.000
	0.4M	1.873 ± 0.159	$+0.023$

Table A24: Equal-budget baseline sensitivity on MEAL Level-1 Easy (10-task average).

Method	Configuration	10-task Avg.	Δ vs. Default
ReDo	$\tau = 0.05$	1.624 ± 0.149	-0.006
	$\tau = \mathbf{0.10}$ (default)	$\mathbf{1.630 \pm 0.090}$	0.000
	$\tau = 0.20$	1.631 ± 0.161	$+0.001$
ReBorn	$\alpha = 0.05$	1.571 ± 0.133	-0.019
	$\alpha = \mathbf{0.10}$ (default)	$\mathbf{1.590 \pm 0.080}$	0.000
	$\alpha = 0.20$	1.577 ± 0.124	-0.013
ReGraMa	$\tau_g = 0.005$	1.710 ± 0.124	0.000
	$\tau_{\mathbf{g}} = \mathbf{0.010}$ (default)	$\mathbf{1.710 \pm 0.080}$	0.000
	$\tau_g = 0.020$	1.728 ± 0.028	$+0.018$

In particular, its advantage over the strongest competing baseline remains 10.36, 7.55, and 8.39 percentage points for 5, 10, and 20 seeds, respectively. This stability indicates that the observed improvement is not attributable to a small number of favorable random initializations.

We further perform a 30-seed evaluation on the sequential one-step matrix game, where the underlying learner exhibits substantially more severe re-adaptation failure. As reported in Table A26, KNIFE progressively reduces the approximation bias to 0.12 ± 0.06 at the final recovery stage, whereas all competing methods retain substantially larger residual errors. The same qualitative behavior therefore persists under a much larger set of independent runs and in a setting with substantially stronger plasticity failure.

Taken together, these two evaluations show that the main empirical conclusions are stable as the number of random seeds increases. The relative advantage of KNIFE persists across both a large-scale

Table A25: Robustness to the number of random seeds on SMACv2 5gen_protoss Task Drift. Results report test win rate (%) with 95% confidence intervals. The relative ranking remains stable as the number of independent seeds increases from 5 to 20.

Seeds	QMIX	KNIFE	ReDo	ReGraMa	ReBorn	S&P	CBP	PI
5	82.75 ± 1.40	$\mathbf{94.68 \pm 0.74}$	79.19 ± 0.63	81.86 ± 0.91	84.17 ± 2.23	81.61 ± 2.53	84.32 ± 0.72	82.27 ± 1.19
10	78.74 ± 4.34	$\mathbf{92.70 \pm 2.00}$	78.34 ± 3.56	84.17 ± 3.71	83.54 ± 1.45	83.96 ± 0.88	85.15 ± 1.24	82.92 ± 3.09
20	79.51 ± 2.33	$\mathbf{93.99 \pm 1.75}$	79.73 ± 1.91	82.41 ± 1.95	82.52 ± 1.91	83.75 ± 1.25	85.60 ± 1.00	84.31 ± 1.62

Table A26: Thirty-seed evaluation on the sequential one-step matrix game. Entries report matrix approximation bias during successive recovery stages after returning to the initial task (lower is better). Results are reported as mean ± 95% confidence interval over 30 independent random seeds.

Method	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
QMIX	19.02 ± 1.27	15.69 ± 2.11	15.53 ± 2.12	15.47 ± 2.13	15.49 ± 2.13	15.49 ± 2.14	14.53 ± 2.24	14.54 ± 2.24	14.54 ± 2.25	14.19 ± 2.21
KNIFE	20.83 ± 1.10	15.18 ± 3.68	14.05 ± 4.05	$\mathbf{7.23 \pm 3.78}$	$\mathbf{2.85 \pm 2.37}$	$\mathbf{0.31 \pm 0.10}$	$\mathbf{0.21 \pm 0.11}$	$\mathbf{0.06 \pm 0.02}$	$\mathbf{0.03 \pm 0.01}$	$\mathbf{0.12 \pm 0.06}$
CBP	15.77 ± 2.14	15.74 ± 2.15	15.52 ± 2.13	14.55 ± 2.23	14.28 ± 2.24	14.23 ± 2.24	14.21 ± 2.24	14.24 ± 2.25	14.22 ± 2.25	14.20 ± 2.24
ReBorn	22.04 ± 0.13	22.02 ± 0.14	22.02 ± 0.14	22.02 ± 0.14	22.03 ± 0.14	21.56 ± 0.47	21.55 ± 0.47	21.54 ± 0.48	21.54 ± 0.46	21.55 ± 0.47
ReDo	21.48 ± 0.40	17.90 ± 1.66	15.82 ± 2.19	15.58 ± 2.24	15.27 ± 2.33	15.21 ± 2.34	14.13 ± 2.41	14.09 ± 2.43	14.10 ± 2.42	14.04 ± 2.43
ReGraMa	17.91 ± 1.89	12.30 ± 2.78	11.84 ± 2.90	11.60 ± 2.95	11.62 ± 2.95	11.60 ± 2.96	11.56 ± 2.97	11.57 ± 2.97	11.58 ± 2.97	11.58 ± 2.96
ReSet	34.18 ± 3.40	14.61 ± 3.00	18.81 ± 4.38	13.89 ± 3.59	10.69 ± 3.23	15.46 ± 4.29	3.44 ± 1.02	15.99 ± 3.54	15.31 ± 3.67	21.09 ± 1.73
S&P	16.33 ± 1.65	12.38 ± 2.03	10.66 ± 2.13	8.52 ± 2.15	7.55 ± 2.11	7.50 ± 2.12	7.47 ± 2.13	7.48 ± 2.12	7.47 ± 2.13	7.45 ± 2.13

MARL benchmark and a controlled matrix-game setting, providing evidence that the reported gains are robust to stochastic variation rather than being driven by a particular choice of random seeds.

E Discussion

E.1 Societal impact

This paper studies continual multi-agent reinforcement learning (MARL) and proposes a neuron-level plasticity injection method to improve adaptation under non-stationary reward profiles. The work is evaluated only on standard benchmark environments. Potential positive impacts include improved robustness and reduced retraining when objectives change. As with many RL techniques, improved adaptation could also be misused in unsafe or unmonitored deployments; we do not claim readiness for real-world use.

E.2 Failure Severity Across Task Streams

The different failure patterns in the sequential matrix game and SMAC streams likely reflect task similarity. The matrix tasks are deliberately constructed with sharply different optima and little positive transfer, so standard learners can fail almost completely to re-adapt. By contrast, the SMAC streams retain reusable combat strategies across tasks, and the same learners typically degrade rather than uniformly collapse. Thus, the two benchmarks expose different severities of impaired adaptation rather than contradictory behaviors. Detailed matrix-game results are provided in Appendix Figure A8 and Table A26.

E.3 Limitations and future work

While we obtain encouraging results with several MARL learners and scale the evaluated setting to 40 agents, our evaluations remain confined to simulated benchmark environments. The non-value-factorization extensions are concentrated on one task stream, and the single-agent evaluation uses one benchmark; broader claims across architectures and continual-RL settings therefore require further validation. The dynamic complexities of real-world physical systems (e.g., severe sim-to-real gaps and unpredictable sensor noise) also remain to be explored. Additionally, while our pruning mechanism ensures long-term memory efficiency, the initial plasticity injection phase introduces a temporary memory overhead. Future work will investigate memory-constant injection mechanisms and evaluate KNIFE on additional task streams, benchmarks, and physical multi-agent robotic systems.

KNIFE also relies on temporarily available neuron capacity during plasticity injection. When no recyclable capacity is available, some detected neurons cannot be immediately intervened on and

must wait for later pruning cycles. Moreover, an injected active neuron may itself become stagnant after it is converted back to a regular neuron, requiring subsequent re-detection and intervention.

KNIFE also introduces method-specific hyperparameters, including the RUA thresholds, the identification-and-injection interval, and the frozen-branch decay factor applied after every learner update. Although a shared configuration works across our evaluated settings, reducing this tuning burden through adaptive thresholds, decay rates, or intervention schedules remains an important direction for future work.

Exhaustively tuning every baseline on each newly constructed task stream would require a substantially larger computational budget. We therefore evaluate all methods under a common no-environment-specific-tuning policy: KNIFE uses one fixed configuration, the baselines use their published method-specific defaults, and all periodic intervention methods share the same execution interval. Our results consequently do not claim the best attainable performance of every baseline, and environment-specific tuning may improve their absolute performance.