MEAL: A Benchmark for Continual Multi-Agent Reinforcement Learning

Tristan Tomilin[1] **Luka van den Boogaard**[1] **Samuel Garcin**[2]
Bram Grooten[1] **Meng Fang**[3,1] **Mykola Pechenizkiy**[1]
[1]Eindhoven University of Technology [2]University of Edinburgh [3]University of Liverpool
{t.tomilin,l.v.d.boogaard,b.grooten,m.pechenizkiy}@tue.nl
s.garcin@ed.ac.uk meng.fang@liverpool.ac.uk

Abstract

Benchmarks play a crucial role in the development and analysis of reinforcement learning (RL) algorithms, with environment availability strongly impacting research. One particularly underexplored intersection is continual learning (CL) in cooperative multi-agent settings. To remedy this, we introduce **MEAL** (**M**ulti-agent **E**nvironments for **A**daptive **L**earning), the first benchmark tailored for continual multi-agent reinforcement learning (CMARL). Existing CL benchmarks run environments on the CPU, leading to computational bottlenecks and limiting the length of task sequences. MEAL leverages JAX for GPU acceleration, enabling continual learning across sequences of 100 tasks on a standard desktop PC in a few hours. We show that naïvely combining popular CL and MARL methods yields strong performance on simple environments, but fails to scale to more complex settings requiring sustained coordination and adaptation. Our ablation study identifies architectural and algorithmic features critical for CMARL on MEAL.

1 Introduction

Continual RL has recently attracted growing interest [17, 9, 10, 14], but remains largely unexplored in multi-agent settings [44, 45]. Combining the two introduces unique challenges. In cooperative environments, agents must establish implicit conventions or roles for effective coordination [38]. As tasks or dynamics shift, these conventions can break down, making continual MARL significantly harder than its single-agent counterpart. Forgetting past partners or roles can cause the entire team to fail, amplifying the impact of catastrophic forgetting through inter-agent dependencies. Unlike traditional MARL, CMARL involves non-stationarity not only due to the presence of other learning agents, but also from an unknown task distribution [45]. This dual pressure demands agents to generalize, adapt, and transfer knowledge more robustly than in standard single-agent continual or static multi-agent settings. This setting is prevalent in applications where agents must adapt to evolving environments without forgetting prior coordination strategies. For instance, autonomous vehicles must navigate unseen roads, adapt to new traffic regulations, and interact with unfamiliar human drivers, while coordinating with other AVs. Similarly, warehouse robots deployed in a new facility must quickly adapt to unseen layouts and workflows, while preserving established collaborative behaviors.

To analyze how current methods handle the interplay between CL and MARL, and to drive progress in this domain, we introduce **MEAL**, the first benchmark for CMARL. To the best of our knowledge, MEAL[1] is also the first continual RL library to leverage JAX for end-to-end GPU acceleration. Traditional CPU-based benchmarks are limited to short sequences (5–15 tasks) due to low environment throughput and task diversity [37, 32, 40], making them ill-suited for the computational demands

[1]The code and environments are accessible on GitHub.

18th European Workshop on Reinforcement Learning (EWRL 2025).

Table 1: Comparison of existing Reinforcement Learning benchmarks with MEAL.

Benchmark	No. Tasks	Difficulty Levels	GPU-accelerated	Action Space	Multi-Agent	Continual Learning
CORA [32]	31	✗	✓	Mixed	✗	✓
MPE [27]	7	✗	✗	Continuous	✓	✗
SMAC [36]	14	✓	✗	Discrete	✓	✗
Continual World [41]	10	✗	✗	Continuous	✗	✓
Melting Pot [2]	49	✗	✗	Discrete	✓	✗
Google Football [23]	14	✓	✓	Discrete	✓	✗
JaxMARL [35]	33	✗	✓	Mixed	✓	✗
COOM [40]	8	✓	✗	Discrete	✗	✓
MEAL	∞	✓	✓	Discrete	✓	✓

of CL across long task sequences. MEAL's end-to-end JAX pipeline removes this barrier, enabling training on up to 100 tasks within a few hours on a single desktop GPU. This unlocks new research directions for scalable, cooperative continual learning in resource-constrained settings.

MEAL is built on Overcooked [7], a widely used cooperative MARL environment [19, 42, 38], providing a strong foundation for benchmarking. Prior work has shown that agents tend to exploit spurious correlations in fixed layouts, resulting in poor generalization even under minor modifications [22]. This makes Overcooked particularly well-suited for continual learning: even small layout variations can present a significant challenge. To succeed across a sequence of such tasks, agents must avoid overfitting to layout-specific behaviors and instead learn coordination strategies that are robust and transferable.

The **contributions** of our work are three-fold. (1) We introduce MEAL, the first CMARL benchmark, consisting of procedurally generated Overcooked environments spanning three difficulty levels. (2) We leverage JAX to build the first end-to-end GPU-accelerated task sequences for continual RL, enabling efficient training on low-budget setups. (3) We implement six popular CL methods in JAX and evaluate them in MEAL, revealing key shortcomings in retaining cooperative behaviors and adapting to shifting roles across tasks.

2 Related Work

Continual Reinforcement Learning (CRL) CRL studies how agents can learn sequentially from a stream of tasks without forgetting previous knowledge. A wide range of methods have been adapted from the CL literature to facilitate the RL setting, including regularization-based approaches such as EWC [21], SI [46], and MAS [3]; architectural strategies such as PackNet [25]; and replay-based methods like RePR [4]. More recent works focus on scalability [17], memory efficiency [10], and stability during training [9]. However, these methods are almost exclusively developed for single-agent settings, and their behavior under multi-agent coordination remains largely unexplored.

Multi-Agent Reinforcement Learning (MARL) In MARL, multiple agents learn to act in a shared environment, often with partial observability and either cooperative or competitive goals [18, 29]. A major focus has been on cooperative settings, where agents share a reward function and must learn to coordinate [24, 15]. Popular algorithms include IPPO [11], VDN [39], QMIX [33], and MAPPO [43]. Many benchmarks assume a static environment and fixed task, making them unsuitable for studying continual learning or transfer across environments.

Benchmarks Standard CRL benchmarks include Continual World [41], COOM [40], and CORA [32]. While effective in single-agent settings, they either lack multi-agent capabilities or suffer from slow CPU-bound environments. For MARL, environments like SMAC [36], MPE [27], and Melting Pot [2] are widely used, but are not designed for continual evaluation. Overcooked [7] has emerged as a useful domain for studying coordination, with recent implementations in JAX [35]. Our benchmark builds on Overcooked and introduces procedural variation to create long task sequences for continual MARL.

Overcooked The Overcooked environment [7] is a cooperative multi-agent benchmark inspired by the popular video game of the same name. Agents control chefs in a grid-based kitchen, coordinating

to prepare and deliver dishes through sequences of interactions with environment objects such as pots, ingredient dispensers, plate stations, and delivery counters. The environment is designed to require both motion and strategy coordination, making it a standard testbed for evaluating collaborative behaviors. Compared to the large state spaces and high agent counts in benchmarks like Melting Pot [2] and SMAC [36], Overcooked operates on small grid-based environments with only two agents. However, its complexity arises not from scale but from credit assignment challenges due to shared rewards, and the need for precise coordination, as agents must execute tightly coupled action sequences to complete tasks successfully [18].

3 Preliminaries

Cooperative Multi-Agent MDP We formulate the setting as a fully observable cooperative multi-agent task, modeled as a Markov game defined by the tuple $\langle N, S, A, P, R, \gamma \rangle$, where N is the number of agents, S is the state space, A^i is the action space of agent i with joint action space $A = A^1 \times \cdots \times A^N$, $P : S \times A \times S \to [0, 1]$ is the transition function, $R : S \times A \times S \to \mathbb{R}$ is a shared reward function, and $\gamma \in [0, 1)$ is the discount factor. In the fully observable setting, each agent receives the full state $s \in S$ at every time step.

Continual MARL We consider a continual MARL setting in which a shared policy $\pi_\theta = \pi^i_{\theta i \in N}$ is learned over a sequence of tasks $\mathcal{T} = \mathcal{M}_1, \ldots, \mathcal{M}_N$, where each $\mathcal{M}_i = \langle N, S_i, A, P_i, R_i, \gamma \rangle$ is a fully observable cooperative Markov game with consistent action and observation spaces. At training phase i, agents interact exclusively with $\mathcal{M}_i$ for a fixed number of iterations Δ, collecting trajectories $\tau_{i,1}, \ldots, \tau_{i,\Delta}$ to update their policy. Past tasks are inaccessible, and no joint training is allowed. The objective is to maximize performance on all tasks in the sequence.

4 MEAL

We introduce MEAL, the first benchmark for CMARL, built on the JaxMARL [35] version of Overcooked. JAX [6] provides just-in-time compilation, automatic differentiation, and vectorization through XLA, enabling high-performance and accelerator-agnostic computation.

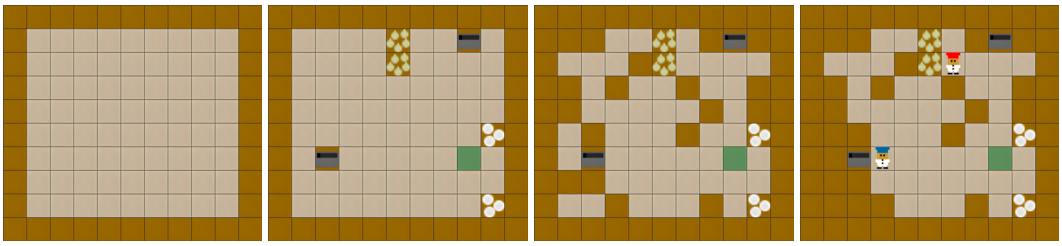
4.1 Environment Specifications

Dynamics Agents act synchronously at each time step. Moves into walls or occupied tiles are no-ops, and simultaneous swaps are disallowed (both agents remain in place). Agents can `interact` with the tile they are facing, which deterministically updates the object's state (pick/place, add onion, plate, deliver). Pots initiate a fixed cook timer of c_{cook}=20 steps when the third onion is added, and the cooked soup can only be plated upon completion.

Observations Each agent receives a fully observable grid-based observation of shape $(H, W, 26)$, where H and W are the height and width of the environment, and the 26 channels encode entity types (e.g., walls, agents, onions, plates, pots, delivery stations) and object states (e.g., cooking progress, held item). To ensure compatibility across environments in a continual learning setting, we fix H_{max} and W_{max} to the largest layout size and pad smaller layouts with walls. Observations are then standardized to the shape $(H_{\text{max}}, W_{\text{max}}, 26)$.

Action Space At each timestep, both agents select one of six discrete actions from a shared action space $\mathcal{A} = \{$`up`, `down`, `left`, `right`, `stay`, `interact`$\}$. Movement actions translate the agent forward if the target tile is free (i.e., not a wall or occupied), while `stay` maintains the current position. The `interact` action is context-dependent and allows agents to pick up or place items, add ingredients to pots, serve completed dishes, or deliver them at the goal location. Importantly, there is no built-in communication action; all coordination emerges from environment interactions.

Rewards Agents receive a shared team reward: $r_t = r_{\text{deliver}} + r_{\text{onion}} \cdot \mathbb{1}_{\{\text{onion_in_pot}\}} + r_{\text{plate}} \cdot \mathbb{1}_{\{\text{plate_pickup}\}} + r_{\text{soup}} \cdot \mathbb{1}_{\{\text{soup_pickup}\}}$, where $r_{\text{deliver}} = 20$ is the reward for delivering soup, and the other terms provide shaped rewards for intermediate progress. We include two reward settings: in the **sparse** setting, $r_{\text{onion}} = r_{\text{plate}} = r_{\text{soup}} = 0$; in the **dense** setting, $r_{\text{onion}} = r_{\text{plate}} = 3$, and $r_{\text{soup}} = 5$.



(a) Empty grid drawn with outer walls. (b) Interactive stations sampled at random locations. (c) Grid filled with walls to match obstacle density. (d) Agents added and unreachable tiles pruned.

Figure 1: Procedural generation pipeline of a **hard** layout. Starting from an empty grid with outer walls, the generator injects interactive stations, adds walls to match the desired obstacle density, places agents, and finally prunes unreachable tiles.

4.2 MEAL Generator

Existing continual RL benchmarks only provide a fixed set of tasks [37, 32, 40]. To avoid over-fitting to a fixed set of environments, we procedurally generate new Overcooked kitchens on the fly. The generator G draws a random width and height from the specified range, places an outer wall, then sequentially injects the interactive tiles (goal, pot, onion pile, plate pile), extra internal walls to match the target obstacle density, and finally, the agents' starting positions. Figure 1 depicts the steps in the pipeline, and the process is described more in-depth in Appendix A.2. Each candidate grid is accepted only if a built-in validation module confirms that both agents can complete at least one cook–deliver cycle. This yields a continuous space of solvable, variable-sized kitchens that we can learn continually. We bring further details about the validator in Appendix A.3. Our approach offers a virtually infinite supply of tasks and evaluates true lifelong learning under continual exposure to unseen configurations. To ensure reproducibility and a fair comparison between methods, the generation process can be fully controlled via a user-specified random seed.

4.3 Layout Difficulty

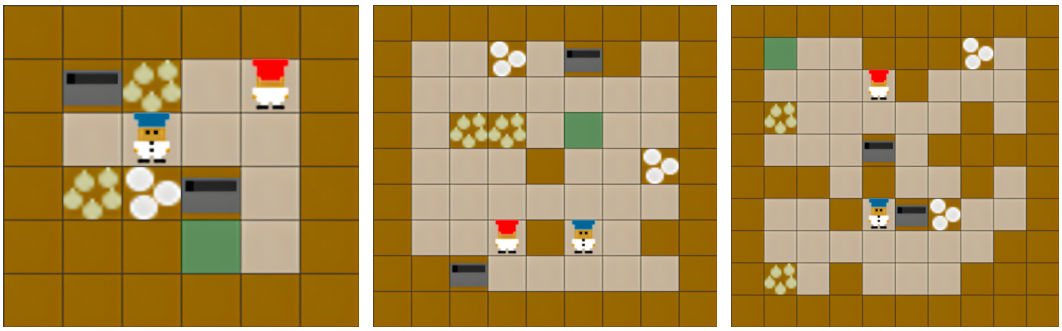
We categorize environment difficulty based on procedurally generated layout characteristics. We vary the (1) grid width, (2) grid height, and (3) obstacle density. This approach produces diverse spatial configurations while maintaining consistent difficulty within each level. Figure 2 depicts layouts of each difficulty. As grid size and the number of impassable tiles increase, agents must develop more sophisticated coordination strategies. Higher difficulty layouts feature longer paths between key items, tighter bottlenecks, and greater structural variability, all of which make exploration, retention, and adaptation more challenging. Level 1 tasks are designed to be simple enough for existing methods to achieve reasonably high scores, enabling better comparisons and behavioral analysis. Higher levels are intended to challenge future methods. Although we currently include three difficulty levels, it is straightforward to extend the framework. Although obstacle density has a practical upper bound, the grid size can be increased arbitrarily to scale up environment complexity.

4.4 Task Sequences

MEAL provides discrete task sequences $\mathcal{T} = (\mathcal{M}_1, \ldots, \mathcal{M}_N)$ rather than a continuous domain shift. For a chosen difficulty level $\ell \in \{1, 2, 3\}$, we sample N solvable layouts i.i.d. from the generator G_ℓ with a fixed seed. At task boundaries, we carry over the optimizer state and policy parameters, reset rollout buffers, and advance the RNG. We explore three sequence regimes: (i) **fixed-level**, where all tasks are drawn from the same difficulty level; (ii) **curriculum**, where sequences contain an equal number of tasks in increasing difficulty level (see Appendix E), and (iii) **repetition**, where a fixed sequence is repeated r times (used in the network plasticity study in Section 5.5).

4.5 Evaluation Metrics

We measure task performance as the number of soups successfully delivered during an episode. Since MEAL layouts vary greatly in size, structure, number of interactive stations, and distances between them, raw delivery counts are not directly comparable. To unify performance across tasks and sequences for continual learning metrics, we divide the delivery count by an upper bound. We



(a) **Level 1** (Easy): $6 \leq$ width/height ≤ 7, obstacle density $\approx 15\%$. Layouts are compact, making exploration easy. Interactable items are close together, making travel distances short. Agents can often complete the task independently with no coordination.

(b) **Level 2** (Medium): $8 \leq$ width/height ≤ 9, obstacle density $\approx 25\%$. Exploration is harder as stations are more spread out. Layouts often introduce chokepoints, requiring agents to coordinate movement and avoid congestion.

(c) **Level 3** (Hard): $10 \leq$ width/height ≤ 11, obstacle density $\approx 35\%$. Layouts are likely to split the map into disjoint regions, forcing agents to specialize. Solving the task requires deliberate cooperation and division of labor.

Figure 2: Representative Overcooked layouts generated at each difficulty level. Increasing grid size and obstacle density lead to longer travel distances, harder exploration, and greater coordination demands.

calculate this bound for each layout by finding the optimal cook-deliver cycle for a single agent (see Appendix A.1), accounting for the shortest paths between onion piles, pots, plate piles, and delivery counters, the fixed cooking time, and pickup/drop interactions. Repeating this cycle over the length of the episode yields a soup-total count. A normalized score of 1 indicates that the agent(s) achieved the optimal performance possible for a single agent, while values above 1 reflect effective cooperation that exceeds solo efficiency. Let $s_i(t)$ denote this normalized delivery score on task i at timestep t. Suppose that the training sequence consists of N tasks, each lasting Δ steps, resulting in a total of $T = N \cdot \Delta$ timesteps. The i-th task is therefore trained during the interval $t \in [(i-1)\Delta, i\Delta]$. Following prior work [41, 40], MEAL measures performance in three continual learning metrics:

Average Normalized Score To capture the balance between stability and plasticity, we report the mean score across all tasks at the end of training:

$$\mathcal{A} = \frac{1}{N} \sum_{i=1}^{N} s_i(T) \tag{1}$$

Forgetting Forgetting quantifies the degradation in performance on past tasks due to interference from training on later ones. We exclude the final task, as no forgetting can occur without further training. For each task $i < N$, we compute the difference between the average score over the final k steps after training on task i and the average over the final k steps of the full sequence T:

$$\mathcal{F} = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\frac{1}{k} \sum_{t=i\Delta-k}^{i\Delta-1} s_i(t) - \frac{1}{k} \sum_{t=T-k}^{T-1} s_i(t) \right) \tag{2}$$

Forward Transfer Forward transfer measures how prior experience accelerates learning of new tasks. Rather than evaluating final performance, it captures how quickly each task is learned relative to a single-task baseline. We first compute the normalized area under the learning curves (AUC) for both the agent and the baseline:

$$\mathrm{AUC}_i = \frac{1}{\Delta} \int_{(i-1)\Delta}^{i\Delta} s_i(t)\, dt, \qquad \mathrm{AUC}_i^{\mathrm{b}} = \frac{1}{\Delta} \int_{0}^{\Delta} s_i^{\mathrm{b}}(t)\, dt. \tag{3}$$

Forward transfer for task i is the area between these curves, positive when prior training helps, and negative when it hinders. We report the average forward transfer across the whole sequence:

$$\mathcal{FT}_i = \frac{\mathrm{AUC}_i - \mathrm{AUC}_i^{\mathrm{b}}}{1 - \mathrm{AUC}_i^{\mathrm{b}}}, \qquad \mathcal{FT} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{FT}_i. \tag{4}$$

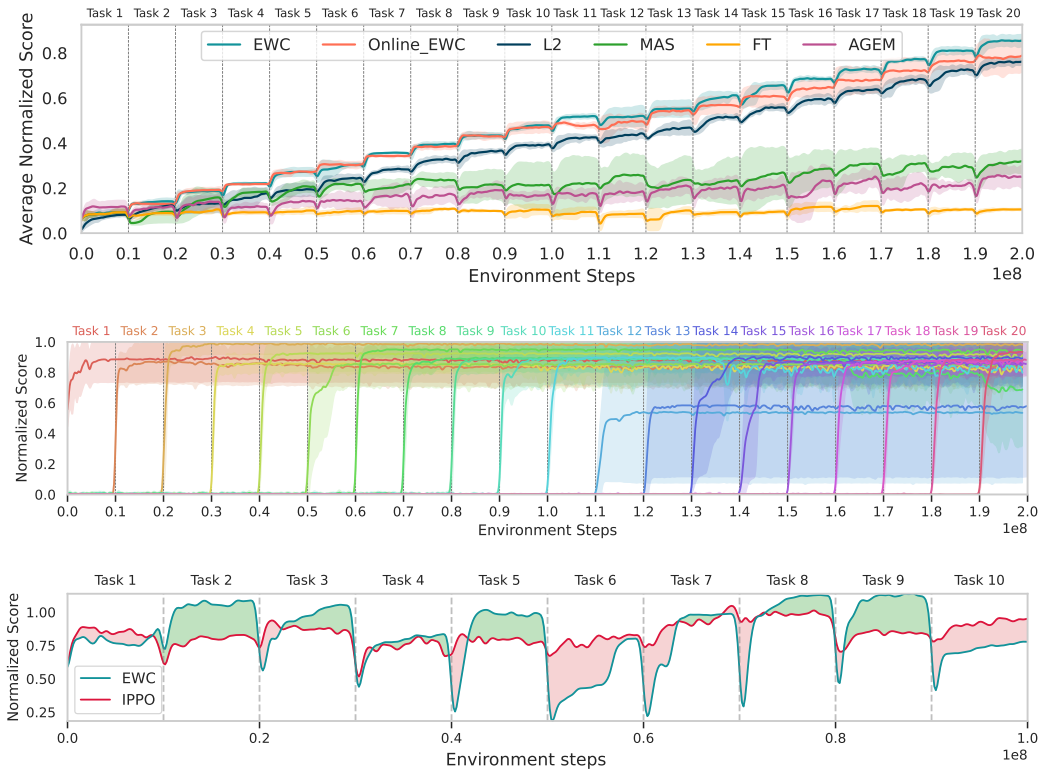


Figure 3: **Top:** The average normalized score evaluation curves on Level 1 tasks show a notable performance gap across baselines. **Middle**: Per-task evaluation scores for EWC on Level 1 indicate near-perfect retention. **Bottom**: EWC manages to outperform the standard IPPO baseline by transferring knowledge forward on most Level 2 tasks, while under-performing in others. The green area between the curves indicates positive forward transfer, while red represents the negative counterpart.

5 Experiments

5.1 Setup

The agent is trained on each task $\mathcal{T}_i$ for $\Delta = 10^7$ environment steps on-policy with the dense reward setting, repeated over five seeds. In our experiments, we adopt the **task-incremental** continual learning paradigm, in which the task identity is known during both training and evaluation. During training, we evaluate the policy after every 100 updates by running 10 evaluation episodes on all tasks in the sequence. The results are displayed with 95% confidence intervals. We leverage JAX to reduce the wall-clock time for training on a single task to around 5 minutes. Out experiments are conducted on a dedicated compute node with a 72-core 3.2 GHz AMD EPYC 7F72 CPU and a single NVIDIA A100 GPU. We adopt many of JaxMARL's default settings for our network configuration, IPPO setup, and training processes. For exact hyperparameters please refer to Appendix B.2.

5.2 Baseline Comparison

We evaluate popular CL methods. Fine-Tuning (**FT**) is a naive baseline where the policy is trained sequentially across tasks without any mechanism to prevent forgetting. **L2-Regularization** [21] adds a penalty on parameter changes to promote stability. **EWC** [21] penalizes changes to important parameters, with importance measured using the Fisher Information Matrix. **Online EWC** is a variant that maintains a running estimate of parameter importance. **MAS** [3] computes importance based on how parameters influence the policy's output, rather than gradients. **AGEM** [8] is a replay-based method that projects the current gradient update to avoid interference with past tasks, using a memory buffer of stored experiences. As the default MARL algorithm, we opt for **IPPO** [11]. It is a natural choice as it can be seamlessly integrated with all model-free CL methods. Moreover, it has been

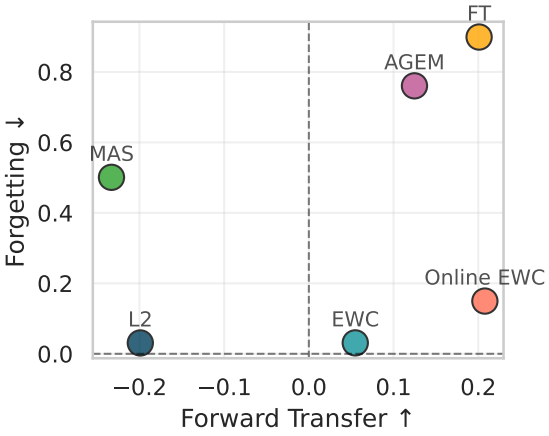


Figure 4: Jointly visualizing forward transfer and forgetting results on Level 1 reveals the classic stability–plasticity trade-off in continual learning.

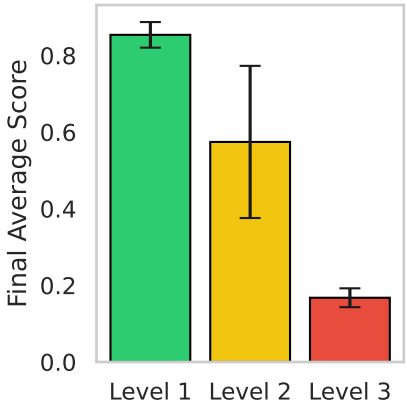


Figure 5: EWC's performance notably declines as layout complexity increases. Most high-level tasks remain unsolved.

Table 2: Baseline comparison results across three difficulty levels. The confidence intervals are omitted for brevity, see Appendix J for the full results.

Method	Level 1			Level 2			Level 3		
	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
FT	0.048	0.899	0.201	0.041	0.790	0.065	0.010	0.519	-0.157
EWC	**0.839**	0.031	0.055	**0.604**	**0.061**	-0.086	0.178	0.053	-0.650
Online EWC	0.769	0.150	**0.208**	0.585	0.211	**0.152**	**0.306**	0.168	**-0.149**
MAS	0.281	0.501	-0.233	0.155	0.423	-0.355	0.034	0.245	-0.542
L2	0.753	**0.031**	-0.199	0.496	0.106	-0.527	0.127	**0.052**	-0.827
AGEM	0.204	0.761	0.125	0.117	0.625	-0.083	0.037	0.457	-0.169

shown to outperform other MARL approaches on SMAC [11] and Overcooked [35], making it a strong candidate for evaluating CMARL in MEAL.

Figure 3 (top) compares our baselines on Level 1, and Table 2 reports the exact metrics for all levels. Fine-Tuning (FT) and AGEM show higher forward transfer, but catastrophically fail at retention: once a task is left behind, performance rapidly collapses. EWC and L2 exhibit near-perfect knowledge retention on all levels, with EWC also ranking highest in average score on lower levels. Figure 3 (middle) visualizes EWC's per-task stability. EWC locks parameters to all past tasks via a cumulative Fisher penalty, which pays off on Levels 1–2, where layouts are small. However, on Level 3, that rigidity bites, as harder layouts demand larger representation shifts, causing EWC to underfit. By contrast, Online EWC uses a decayed Fisher that down-weights older tasks and manages to keep enough plasticity to learn the new layouts (higher $\mathcal{A}$ and $\mathcal{FT}$). See Appendix C for a deeper analysis of this comparison. MAS performs poorly in all metrics, although outperforming FT and AGEM. Notably, the simplistic difficulty level design of MEAL presents notable challenges, as EWC's score diminishes with increasing difficulty (Figure 5). Figure 4 illustrates the fundamental stability-plasticity trade-off in CL. L2 achieves excellent retention but limited forward transfer, while Fine-Tuning and AGEM demonstrate high plasticity with severe forgetting. EWC and its online version provide a middle ground, balancing both objectives more effectively than other approaches.

5.3 Ablation Study

To determine which components are crucial for CMARL on MEAL, we ablate five components in our default IPPO learning setup: multi-head architectures, task identity inputs, critic regularization, layer normalization, and replacing the MLP with a CNN encoder. The results in Figure 6 reveal that multi-head outputs are most critical for MEAL task sequences. Removing them consistently devastates performance across all methods, likely due to uncontrolled interference between tasks

Table 3: Comparison of EWC with PPO/IPPO across 1–3 agent task sequences. Two agents yield the best results due to parallelism and cooperative potential. Adding a third agent introduces instability, non-stationarity, and coordination challenges that hurt performance.

Agents	Level 1			Level 2			Level 3		
	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
1 Agent	0.622	0.046	-0.045	0.343	0.071	-0.458	**0.285**	0.159	**-0.531**
2 Agents	**0.839**	**0.031**	**0.055**	**0.604**	**0.061**	**-0.086**	0.178	**0.053**	-0.650
3 Agents	0.476	0.125	-0.676	0.277	0.132	-0.896	0.117	0.094	-0.978

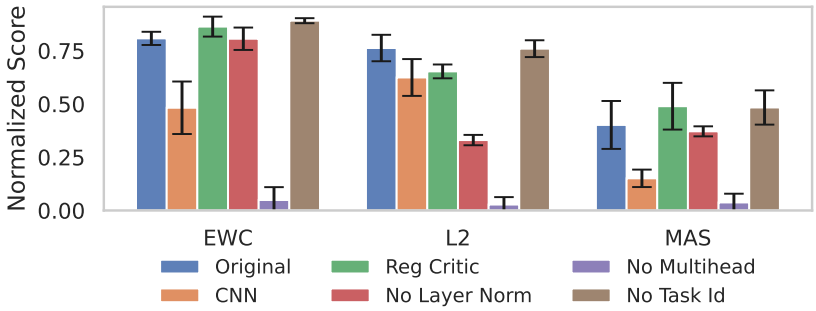


Figure 6: **Ablation results on Level 1 10-task sequences**. Removing the multi-head architecture leads to severe performance degradation across all methods. Including the task ID and critic regularization has a negligible effect. Layer normalization has little impact on EWC or MAS, but significantly improves L2. A CNN encoder yields worse results than a simple MLP variant.

in the shared output head. In contrast, not providing the model with the one-hot encoded task ID vector has a negligible effect. Prior continual RL studies [41, 40] report that it is beneficial to only regularize the actor and let the critic adapt freely. In our experiments, however, we find that this has little effect. Layer normalization shows method-specific sensitivity: while it makes little difference for EWC and MAS, it more than doubles the performance of L2 regularization. This is likely because L2 penalizes absolute weight magnitudes, and layer norm helps stabilize activations across tasks, mitigating harmful scale drift. Finally, swapping to a CNN encoder substantially hurts performance for all methods. Given the small layouts in Level 1 tasks (6×6 to 7×7), CNNs struggle to extract meaningful features and add unnecessary parameter overhead, making simple MLPs the better fit in this setting.

5.4 N-Agent MEAL

To better analyze the multi-agent dimension of CMARL, we extend MEAL to support an N-agent setting, allowing us to systematically study how the number of cooperating agents affects continual learning. We run EWC combined with PPO for the single-agent case, and IPPO for 2 and 3 agents.

A single agent delivers less soup than two because it cannot parallelize tasks: while one agent delivers soup, the other can already load the pot with onions. However, in Level 3, the single-agent PPO outperforms the two-agent IPPO. We observed that, under the CL setting, the two-agent setup solved fewer tasks due to the larger grid size in harder layouts: increasing both the observation space and the number of agents makes the learning problem more complex for IPPO, which struggles to learn a good policy.

Adding a third agent further hurts performance for similar reasons. IPPO trains independent policies while the environment remains a joint MDP, where transitions and rewards depend on the combined actions of all agents. Moving from 2→3 agents expands the joint action space and interaction patterns, amplifies non-stationarity (as two teammates' policies change simultaneously), and makes credit assignment more difficult (since the reward is shared, IPPO does not know which agent made a good action). Without explicit communication or role allocation, IPPO struggles to learn continually as the team and layout size grow. We explore common pitfalls of agent behaviour in Appendix I.

Table 4: Averaged plasticity metrics for FT on Level 1 task sequences.

Repeats	AUCL ↓	FPR ↑	RAUC ↑
1	0.000	1.000	1.000
3	0.154	0.906	0.905
10	0.215	0.866	0.846

Table 5: Effect of reward design on EWC over 20-task Level-1 sequences.

Rewards	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
Shared	**0.90** ± **0.04**	**0.01** ± **0.01**	**0.20** ± **0.08**
Individual	0.84 ± 0.08	0.08 ± 0.07	0.12 ± 0.06
Sparse	0.19 ± 0.04	0.02 ± 0.01	−0.79 ± 0.10

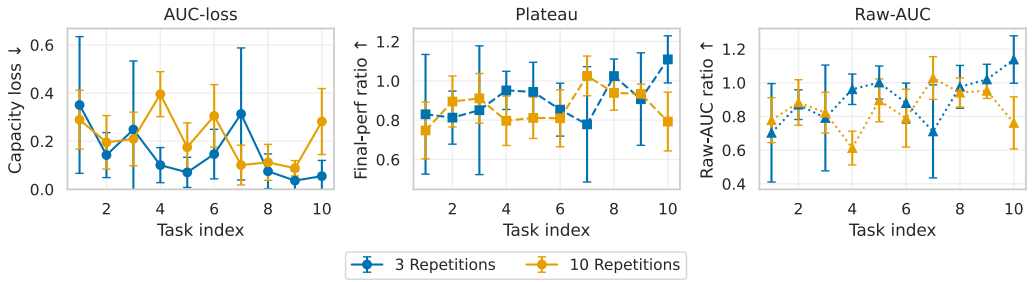


Figure 7: **Plasticity loss in MEAL under fine-tuning (FT).** We measure capacity loss (left), final-performance ratio (middle), and raw-AUC ratio (right) over 10-task Level 1 sequences. Increasing the repetition count from 3 to 10 amplifies degradation across all metrics, indicating reduced plasticity as learning progresses. The high error bars stem from the agent learning a task during one repetition but performing poorly during another. This effect is more pronounced at lower repetitions.

5.5 Network Plasticity

A well-documented pitfall in continual RL is the gradual loss of **plasticity**, an agent's ability to fit new data after many tasks [1, 12]. To test whether MEAL exhibits the same pathology, we repeat a Level 1 10-task sequence multiple times and compare performance between repetitions. We evaluate fine-tuning (FT), as it is the simplest and most cost-effective baseline ranking high in plasticity (Figure 4). We track three standard metrics, each normalized to the initial repetition: (i) **AUC-loss** ↓ captures capacity drop, (ii) Final-performance ratio (**FPR**) ↑ compares the final plateau to the first repetition, and (iii) Raw-AUC ratio (**RAUC**) ↑ measures total reward accumulated during the repeat. For formal definitions of the metrics and training curves, see Appendix G.

We observe that all metrics deteriorate with longer training (Table 4 and Figure 7), confirming that loss of plasticity also appears in the multi-agent setting. Notably, the larger drop in performance occurs between 1 and 3 repetitions, suggesting that early degradation is more severe. AUC-loss increases by roughly 40% when going from 3→10 repetitions. Despite our setting spanning over 1B environment steps, well beyond the scale of prior studies [1, 12], those works report a much stronger loss of plasticity than observed in MEAL. We hypothesize that this difference stems from our experiments using multiple output heads, which isolate task-specific outputs, reduce gradient interference, and preserve prior policies while allowing the backbone to learn transferable features.

5.6 Reward Design

By default, Overcooked agents receive *shared* rewards. We compare this with an *individual* reward mode, where each agent is rewarded solely for its own actions, often leading to greedier behavior and weaker coordination [30, 16, 20]. We also evaluate the *sparse* shared reward setting described in Section 4.1, assessing all three using EWC on 20-task Level 1 sequences.

As shown in Table 5, shared rewards lead to the best performance, although on some tasks, the individual reward setting allows the agents to find a better global solution due to the inherent competitiveness. Agents are motivated to use different onion piles and pots to maximize their own rewards, which often leads to a more efficient solution. However, this competitive drive occasionally prevents them from converging on a stable solution. The sparse reward setting grants rewards only for successful deliveries. Without a targeted exploration mechanism, agents are unlikely to discover a full delivery sequence through random actions even on Level 1 layouts, leading to worse performance.

5.7 Partial Observability

Although Overcooked is fully observable by design, we introduce a partially observable variant to better reflect real-world sensing constraints (limited field of view, occlusions). Following popular MARL environments [34, 26, 2, 13], each agent receives an egocentric, direction-aware observation window with all outside tiles masked. The specification and difficulty scaling of this window are detailed in Appendix D.

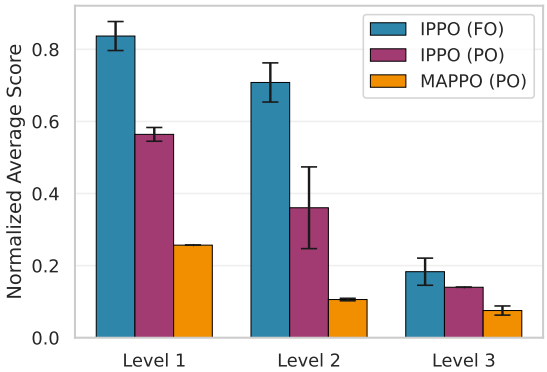


Figure 8: Performance of EWC under full (FO) and partial observability (PO) on 20-task Level 1 sequences. FO yields the best results overall. MAPPO underperforms in the continual learning setting.

In this setting, MAPPO [43] is known to outperform IPPO, since its centralized critic can 1) more accurately estimate individual contributions to shared rewards under partial observability, and 2) reduce non-stationarity by conditioning value estimates on the joint actions of all agents, leading to more stable and coordinated policy updates. We investigate this by running a 20-task sequence under partial observability (PO) with EWC and compare the results with the fully observable (FO) baseline.

Across all levels, IPPO under full observability (FO) clearly dominates the partial setting (PO), and both outperform MAPPO in our CL regime (Figure 8). The gap between IPPO(FO) and IPPO(PO) stems from full state information simplifying credit assignment and stabilizing value targets. Partial observability thus increases task difficulty. Contrary to expectation, MAPPO underperforms IPPO. A plausible cause is a mismatch between MAPPO's centralized critic and the CL regime. Conditioning on joint observations and actions drifts substantially across tasks, yielding noisier targets and stronger cross-task interference, while IPPO's independent critics learn simpler task-local value functions that transfer more stably. These results motivate including PO variants in MEAL to stress coordination under incomplete information for more realistic continual MARL benchmarking.

6 Conclusion

We introduced **MEAL**, a scalable benchmark for continual multi-agent RL, built on JAX for efficient GPU training. The on-demand creation of procedurally generated Overcooked layouts enables long-horizon CMARL studies with controlled difficulty, observability, and agent count. We evaluated combinations of popular CL methods and MARL algorithms, revealing that existing techniques struggle to retain cooperative behaviors while maintaining adaptability to new tasks. The N-agent extension increases coordination demands and exacerbates interference, yielding a harder, more variable task distribution. Partial observability compounds this difficulty, as centralized critics exhibit stronger cross-task drift and interference. Individual rewards weaken coordination and induce negative transfer. A simple curriculum boosts performance on complex layouts under an equal data budget. Training on long task sequences degrades network plasticity in MARL, while multi-head architectures yield the largest structural gains for performance. Our findings suggest that MEAL exposes the dual challenge of cooperation and non-stationarity in CMARL. We see immediate headroom for methods that (i) are purpose-built for CMARL, jointly handling partner and environment-level non-stationarity, (ii) stabilize credit assignment under partial observability across task sequences, and (iii) drive structured exploration and robust coordination in diverse, long-horizon settings. We hope MEAL serves as a solid foundation for pushing this line of work forward.

7 Limitations

While MEAL provides a scalable and diverse testbed for CMARL, several limitations remain. First, MEAL is restricted to discrete action spaces, limiting its applicability. Second, while layout diversity is high, the domain itself is narrow. Overcooked dynamics do not capture the full complexity of real-world multi-agent interactions involving language, negotiation, or long-horizon planning. Third, our benchmark only evaluates task-incremental learning by changing layouts. Future work could extend MEAL to other CL settings. Finally, we only consider CL in settings where the environment layout changes across tasks, but not the partner agent nor environment dynamics.

Acknowledgments

This work was conducted with the assistance of the Dutch national e-infrastructure, generously supported by the SURF Cooperative under grant EINF-12816. Additionally, this research was partially funded by the KOIOS project under grant agreement 101103770.

References

[1] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. In *Conference on lifelong learning agents*, pages 620–636. PMLR, 2023.

[2] John P Agapiou, Alexander Sasha Vezhnevets, Edgar A Duéñez-Guzmán, Jayd Matyas, Yiran Mao, Peter Sunehag, Raphael Köster, Udari Madhushani, Kavya Kopparapu, Ramona Comanescu, et al. Melting pot 2.0. *arXiv preprint arXiv:2211.13746*, 2022.

[3] 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)*, pages 139–154, 2018.

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

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

[7] Micah Carroll, Rohin Shah, Mark K. Ho, Thomas Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/f5b1b89d3db40d65b49f8f9e383ac5dd-Paper.pdf`.

[8] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[9] Feng Chen, Fuguang Han, Cong Guan, Lei Yuan, Zhilong Zhang, Yang Yu, and Zongzhang Zhang. Stable continual reinforcement learning via diffusion-based trajectory replay. *arXiv preprint arXiv:2411.10809*, 2024.

[10] Wesley Chung, Lynn Cherif, Doina Precup, and David Meger. Parseval regularization for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 127937–127967, 2024.

[11] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[12] 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.

[13] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36:37567–37593, 2023.

[14] Zeki Doruk Erden, Donia Gasmi, and Boi Faltings. Continual reinforcement learning via autoencoder-driven task and new environment recognition. In *The Seventeenth Workshop on Adaptive and Learning Agents*.

[15] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[16] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[17] Muhammad Burhan Hafez and Kerim Erekmen. Continual deep reinforcement learning with task-agnostic policy distillation. *Scientific Reports*, 14(1):31661, 2024.

[18] 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.

[19] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pages 4399–4410. PMLR, 2020.

[20] Edward Hughes, Joel Z Leibo, Matthew Phillips, Karl Tuyls, Edgar Dueñez-Guzman, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin McKee, Raphael Koster, et al. Inequity aversion improves cooperation in intertemporal social dilemmas. *Advances in neural information processing systems*, 31, 2018.

[21] 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.

[22] Paul Knott, Micah Carroll, Sam Devlin, Kamil Ciosek, Katja Hofmann, Anca D Dragan, and Rohin Shah. Evaluating the robustness of collaborative agents. *arXiv preprint arXiv:2101.05507*, 2021.

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

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

[25] 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*, pages 7765–7773, 2018.

[26] Sharada Mohanty, Erik Nygren, Florian Laurent, Manuel Schneider, Christian Scheller, Nilabha Bhattacharya, Jeremy Watson, Adrian Egli, Christian Eichenberger, Christian Baumberger, et al. Flatland-rl: Multi-agent reinforcement learning on trains. *arXiv preprint arXiv:2012.05893*, 2020.

[27] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[28] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[29] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *arXiv preprint arXiv:1908.03963*, 2019.

[30] Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. *Advances in neural information processing systems*, 30, 2017.

[31] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.

[32] Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Gupta. Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *Conference on Lifelong Learning Agents*, pages 705–743. PMLR, 2022.

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

[34] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

[35] Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments and algorithms in jax. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pages 2444–2446, 2024.

[36] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[37] Artyom Y Sorokin and Mikhail S Burtsev. Continual and multi-task reinforcement learning with shared episodic memory. *arXiv preprint arXiv:1905.02662*, 2019.

[38] DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34: 14502–14515, 2021.

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

[40] Tristan Tomilin, Meng Fang, Yudi Zhang, and Mykola Pechenizkiy. Coom: a game benchmark for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023.

[41] Maciej Wołczyk, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 34:28496–28510, 2021.

[42] Sarah A Wu, Rose E Wang, James A Evans, Joshua B Tenenbaum, David C Parkes, and Max Kleiman-Weiner. Too many cooks: Bayesian inference for coordinating multi-agent collaboration. *Topics in Cognitive Science*, 13(2):414–432, 2021.

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

[44] Lei Yuan, Ziqian Zhang, Lihe Li, Cong Guan, and Yang Yu. A survey of progress on cooperative multi-agent reinforcement learning in open environment. *arXiv preprint arXiv:2312.01058*, 2023.

[45] Lei Yuan, Lihe Li, Ziqian Zhang, Fuxiang Zhang, Cong Guan, and Yang Yu. Multiagent continual coordination via progressive task contextualization. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

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

A Implementation Details

A.1 Maximum Soup Delivery Calculator

Let a kitchen layout $\mathcal{L}$ be defined by four disjoint sets of tiles (onion piles $\mathcal{O}$, plate piles $\mathcal{P}$, pots $\mathcal{K}$, delivery counters $\mathcal{G}$) and a set of walls $\mathcal{W}$. A tile (x, y) is *walkable* if $(x, y) \notin \mathcal{W}$.

Neighbourhood of an object family. We denote the set of walkable tiles adjacent (in the 4-neighbour sense) to any object in $\mathcal{S}$ as:

$$\mathcal{N}(\mathcal{S}) \;=\; \big\{(x', y') \mid (x, y) \in \mathcal{S},\ \|(x', y') - (x, y)\|_1 = 1,\ (x', y') \notin \mathcal{W}\big\}$$

Shortest obstacle-aware distance. Given two tile sets $A, B \subseteq \mathbb{Z}^2$, we define

$$d(A, B) \;=\; \min_{a \in A,\, b \in B} \mathrm{dist}^{\mathcal{G}_{\mathcal{L}}}_{\text{manhattan}}(a, b),$$

where $\mathcal{G}_{\mathcal{L}}$ is the grid graph induced by walkable tiles. We realize this via a breadth-first search (BFS).

Single-agent cook–deliver cycle. A soup requires three onions, one plate pick-up, one soup pick-up, and one delivery. Let

$$d_{\text{onion}} = d\big(\mathcal{N}(\mathcal{O}), \mathcal{N}(\mathcal{K})\big), \quad d_{\text{plate}} = d\big(\mathcal{N}(\mathcal{P}), \mathcal{N}(\mathcal{K})\big), \quad d_{\text{goal}} = d\big(\mathcal{N}(\mathcal{K}), \mathcal{N}(\mathcal{G})\big).$$

The optimistic *movement cost* for one cycle is

$$c_{\text{move}} = \; 3\, d_{\text{onion}} + d_{\text{plate}} + 1 + d_{\text{goal}} + 3.$$

Interaction overhead. Every pick-up or drop is assumed to take a constant $c_{\text{act}} = 2$ steps (turn + interact). With $n_{\text{int}} = 3{\times}2{+}1{+}1{+}1 = 9$ interactions per cycle, the overhead is $c_{\text{over}} = n_{\text{int}}\, c_{\text{act}} = 18$.

Cycle time and upper bound. Including the fixed cooking time $c_{\text{cook}} = 20$ steps, the single-agent cycle time is

$$T_{\text{cycle}} \;=\; c_{\text{move}} + c_{\text{cook}} + c_{\text{over}}.$$

For an episode horizon H, we upper-bound the number of soups by

$$N_{\max}(\mathcal{L}, H) \;=\; \big\lfloor H / T_{\text{cycle}} \big\rfloor,$$

and convert it to reward with $r_{\text{deliver}} = 20$:

$$R_{\max}(\mathcal{L}, H) \;=\; 20\, N_{\max}(\mathcal{L}, H).$$

The bound assumes *a single agent acting optimally*. It ignores multi–agent collaboration and therefore *underestimates* throughput in layouts where multiple agents can parallelize the workflow. Listing 1 contains the exact implementation.

A.2 Procedural Kitchen Generator

Objective. Given a random seed and user-selectable parameters (number of agents n_a, layout height range $[h_{\min}, h_{\max}]$, layout width range $[w_{\min}, w_{\max}]$, and wall-density ρ), the goal is to emit a *solvable* grid string G representing the `Overcooked` environment.

A.2.1 Notation

Let $h,\ w \sim \mathrm{UniformInt}(h_{\min}, h_{\max}),\ \mathrm{UniformInt}(w_{\min}, w_{\max})$, and denote by $\mathcal{C} = \{(i, j) \mid 1 \le i \le h-2,\ 1 \le j \le w-2\}$ the set of *internal* cells (outer walls excluded). Its cardinality is $N_{\text{int}} = (h-2)(w-2)$. An *unpassable* cell contains either a hard wall (`#`) or an interactive tile; we write $N_{\text{unpass}}(G)$ for the number of such cells in G.

Listing 1 Heuristic upper bound (`calculate_max_soup`).

```
# overcooked_upper_bound.py      (excerpt)
COOK_TIME = 20
ACTION_OVERHEAD = 2
INTERACTIONS_PER_CYCLE = 3 * 2 + 1 + 1 + 1
OVERHEAD_PER_CYCLE = INTERACTIONS_PER_CYCLE * ACTION_OVERHEAD

def calculate_cycle_time(layout, n_agents=2):
    ...
    move_cost = 3 * d_onion + d_plate + 1 + d_goal + 3
    return move_cost + COOK_TIME + OVERHEAD_PER_CYCLE

def calculate_max_soup(layout, episode_len, n_agents=2):
    cyc = calculate_cycle_time(layout, n_agents)
    soups = episode_len // cyc
    return int(soups)
```

A.2.2 Algorithm

The generator performs the following loop until a valid grid is produced (Listing 2):

1. **Draw size.** Sample h, w and create an $h \times w$ matrix initialised to FLOOR tiles, then overwrite the border with WALL.
2. **Place interactive tiles.** For each symbol in {GOAL, POT, ONION_PILE, PLATE_PILE} choose a random multiplicity $m \in \{1, 2\}$ and stamp the symbol onto m uniformly chosen floor cells.
3. **Inject extra walls.** Let $n_{\text{target}} = \lceil \rho\, N_{\text{int}} \rceil$ and $n_{\text{add}} = \max\big(0,\, n_{\text{target}} - N_{\text{unpass}}(G)\big)$. Place n_{add} additional walls on random floor cells.
4. **Place agents.** Stamp n_a AGENT symbols on random remaining floor cells.
5. **Validate.** Run the deterministic `evaluate_grid` solver; if it returns `True`, terminate and return (G), otherwise restart.
6. **Cleanup.** Remove any interactive elements and tiles that are unreachable from all agent positions.
7. **Return.** Output the final grid.

Solvability criterion. The validator (Appendix A.3) checks (i) path connectivity between every agent and each interactive tile family, (ii) at least one pot reachable from an onion pile and a plate pile, and (iii) at least one goal reachable from a pot. This is implemented via multiple breadth-first searches. Appendix A.3 further details the evaluator logic.

Wall-density effect. Because interactive tiles themselves count as obstacles, the algorithm first places them, then *only as many extra walls as needed* to reach the prescribed obstacle ratio ρ. This keeps difficulty roughly constant even when two copies of every station are spawned.

Failure handling. If any placement stage exhausts the pool of empty cells, or the validator rejects the grid, the attempt is aborted and restarted with a fresh h, w sample. We cap retries at `max_attempts` (default 2000); empirically fewer than five attempts suffice for $\rho \leq 0.3$.

Complexity. All placement operations are $O(hw)$ in the worst case (linear scans to collect empty cells), while validation runs a constant number of BFS passes, each $O(hw)$. Hence one successful attempt is $O(hw)$.

Listing 2 Overcooked Layout Generator

```
def generate_random_layout(seed, params):
    rng = random.Random(seed)
    for attempt in range(params.max_attempts):
        h = rng.randint(*params.h_range)
        w = rng.randint(*params.w_range)
        grid = init_floor_with_border(h, w)

        # 1. Interactive tiles
        for sym in [GOAL, POT, ONION_PILE, PLATE_PILE]:
            if not place_random(grid, sym, rng.randint(1, 2), rng):
                break  # restart

        # 2. Extra walls to hit density
        n_target = round(params.wall_density * (h-2)*(w-2))
        n_add = n_target - count_unpassable(grid)
        if not place_random(grid, WALL, n_add, rng):
            continue  # restart

        # 3. Agents
        if not place_random(grid, AGENT, params.n_agents, rng):
            continue

        # 4. Validate
        if evaluate_grid(to_string(grid)):
            return to_string(grid)
```

A.3 Layout Validator

We guarantee that every procedurally generated kitchen is *playable* by running a deterministic validator before training begins. The validator implements ten checks, ranging from basic grid sanity to cooperative reachability. A grid is accepted only if **all** checks pass.

Notation. Let G be an $h \times w$ character matrix with symbols $\{\mathtt{W}, \mathtt{X}, \mathtt{O}, \mathtt{B}, \mathtt{P}, \mathtt{A}, \}$ for walls, delivery, onion pile, plate pile, pot, agent, and floor. Interactive tiles are $\mathcal{I} = \{\mathtt{X}, \mathtt{O}, \mathtt{B}, \mathtt{P}\}$, and unpassable tiles $\mathcal{U} = \mathcal{I} \cup \{\mathtt{W}\}$.

Validation rules.

R1 *Rectangularity* – all rows have equal length.

R2 *Required symbols* – each of W,X,O,B,P,A appears at least once.

R3 *Border integrity* – every outer-row/column tile is in $\{\mathtt{W}\} \cup \mathcal{I}$.

R4 *Interactivity access* – every tile in $\mathcal{I} \cup \{\mathtt{A}\}$ has at least one 4-neighbour that is A or floor.

R5 *Reachable onions* – at least one onion pile is reachable by some agent.

R6 *Usable pots* – at least one pot is reachable *and* lies in the same connected component as a reachable onion.

R7 *Usable delivery* – at least one delivery tile is reachable *and* lies in a component with a usable pot.

R8 *Agent usefulness* – each agent can either interact with an object directly or participate in a hand-off (adjacent wall shared with the other agent's region).

R9 *Coverage* – the union of agents' reachable regions touches every object family in $\mathcal{I}$.

R10 *Handoff counter* – if one agent cannot reach all families, a wall tile adjacent to *both* regions exists, enabling item transfer.

Rules R5–R10 rely on two depth-first searches (DFS) from the agent positions. The DFS explores floor and agent tiles only; whenever it touches an interactive tile, that family is marked as "found." Let $\mathrm{Reach}_k \subseteq [h] \times [w]$ denote tiles reached from agent k ($k \in \{1, 2\}$).

Algorithmic outline. Listing 3 shows a condensed version of the validator.

Listing 3 Condensed Layout Validator.

```
def validate(grid_str):
    g = [list(r) for r in grid_str.splitlines()]
    h, w = len(g), len(g[0])

    # R1-R3 omitted for brevity ...

    # Depth-first search from a start cell
    def dfs(i, j, seen):
        if (i, j) in seen or g[i][j] in UNPASSABLE_TILES - {AGENT}:
            return
        seen.add((i, j))
        for di, dj in ((1,0),(-1,0),(0,1),(0,-1)):
            dfs(i+di, j+dj, seen)

    # Agents and family reachability
    a1, a2 = [(i, j) for i,r in enumerate(g)
                      for j,c in enumerate(r) if c == AGENT]
    reach1, reach2 = set(), set()
    dfs(*a1, reach1); dfs(*a2, reach2)

    # Helper: reachable(\mathcal{S}, reach)
    def any_reach(symbols, reach):
        return any(g[i][j] in symbols for i,j in reach)

    # R5-R7
    if not any_reach({ONION_PILE}, reach1|reach2):          return False
    if not any_reach({POT}, reach1|reach2):                 return False
    if not any_reach({GOAL}, reach1|reach2):                return False

    # R8-R10 (usefulness & hand-off)
    def useful(reach_me, reach_other):
        # direct or shared-wall hand-off
        for i,j in reach_me:
            if g[i][j] in INTERACTIVE_TILES: return True
            if g[i][j] == FLOOR and any(
                (abs(i-i2)+abs(j-j2) == 1 and g[i2][j2] == WALL)
                for i2,j2 in reach_other):
                return True
        return False

    if not useful(reach1, reach2): return False
    if not useful(reach2, reach1): return False
    return True
```

Complexity. All checks are $O(hw)$ and require only two DFS traversals, thus one validation runs in time linear to the grid area and is negligible compared with policy learning.

Practical impact. In practice, fewer than 1% of generator attempts fail validation when wall-density $\rho \leq 0.15$ and kitchen size $\geq 8 \times 8$. We therefore cap retries at 2000 without noticeable overhead.

B Experimental Setup

B.1 Network Architecture

All agents share the same actor–critic backbone, implemented in `Flax`. Two encoder variants are provided:

- **MLP** (default) : observation tensor is flattened to a vector and passed through 2 fully-connected layers of width 128.
- **CNN** : three 32-channel convolutions with kernel sizes 5×5, 3×3, 3×3 feed a 64-unit projection, followed by a single 128-unit dense layer.

Common design knobs (controlled from the CLI) are:

- **Activation** (`relu` *vs.* `tanh`).
- **LayerNorm** : applied after every hidden layer when `use_layer_norm` is enabled.
- **Shared vs. Separate encoder** : with `shared_backbone` the two heads operate on a common representation; otherwise actor and critic keep independent trunks.
- **Multi-head outputs** : if `use_multihead` is set, each head holds a distinct slice of logits/values for every task (`num_tasks` $= |\mathcal{T}|$). The correct slice is selected with the cheap tensor reshape in `choose_head`.
- **Task-one-hot conditioning** : setting `use_task_id` concatenates a one-hot vector of length $|\mathcal{T}|$ before the actor/critic heads, mimicking "oracle" task identifiers used in many CL papers.

All linear/conv layers use orthogonal weight initialisation with gain $\sqrt{2}$ (or 0.01 for policy logits) and zero biases. The policy outputs a `distrax.Categorical`; the critic outputs a scalar.

B.2 Hyperparameters

Table 6 lists settings that are *constant* across every experiment unless stated otherwise. Values match the `Config` dataclass in the training script.

Table 6: Fixed hyper-parameters. All experiments use dense reward shaping, two agents, and IPPO unless noted. CL coefficients λ refer to the regularization strength passed to each method.

Parameter	**Value**
Optimization (IPPO)	
Activation	ReLU
Optimizer	Adam (Optax)
Adam (β_1, β_2)	$(0.9,\ 0.999)$
Adam ϵ	10^{-5}
Weight decay	none
Learning rate η	3×10^{-4}
LR annealing	linear ($3 \times 10^{-4} \to 0$)
Env. steps per task Δ	10^7
Parallel envs	16
Rollout length T	128
Update epochs	8
Minibatches / update	8
Effective batch size	$16 \times 128 = 2048$
Discount γ	0.99
GAE λ	0.957
PPO clip ϵ	0.2
Entropy coef. α_{ent}	0.01
Value-loss coef. α_{vf}	0.5
Max grad-norm	0.5
Continual-learning specifics	
Sequence length $\lvert\mathcal{T}\rvert$	20 (base sequence), repeated r times
Reg. coefficient λ	10^{11} (EWC), 10^9 (MAS), 10^7 (L2)
EWC decay	0.9
Importance episodes / steps	5 / 500
Regularize critic / heads	No / No
AGEM Memory size	100 000 transitions
AGEM Sample size (per proj.)	1024
Miscellaneous	
Reward shaping horizon	2.5×10^6 steps (linear to 0)
Evaluation interval	every 100 updates (10 episodes)
Random seeds	$\{1 .. 5\}$

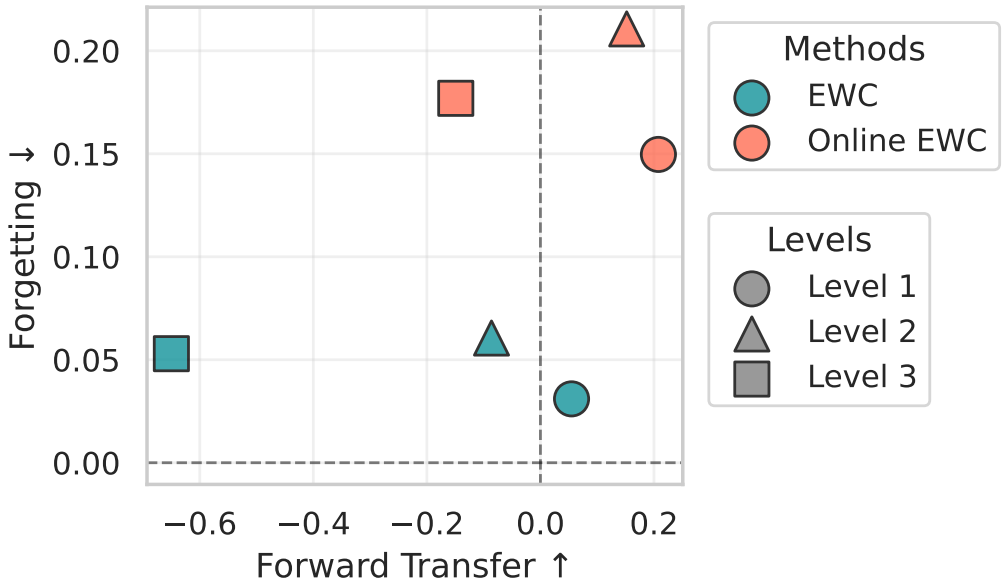


Figure 9: Comparison of EWC and Online EWC across all difficulty levels in terms of forward transfer and forgetting. Each point denotes a method's performance at a given level. Online EWC consistently exhibits higher plasticity (less-negative or positive, particularly at Level 3, while EWC achieves notably lower forgetting on all levels.

C EWC vs. Online EWC

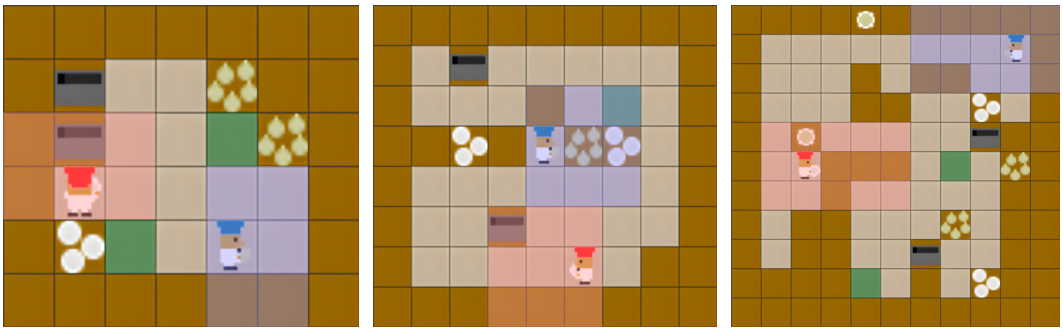
EWC accumulates importance over *all* past tasks and penalizes drift along high-Fisher directions with a fixed quadratic. Online EWC maintains a *running*, exponentially decayed Fisher, emphasizing recent tasks and relaxing old constraints. Both use the same heads, meaning that the penalty acts on the shared trunk. When layouts are small, not only are the tasks easier to learn, but the same features are more likely to work across tasks. Strong anchoring preserves those features, curbing forgetting and yielding a higher average score. The stability–plasticity trade-off is favorable because plasticity demands are modest. This trade-off is visualized in Figure 9, where Online EWC demonstrates higher plasticity at the cost of increased forgetting, while EWC excels in stability but struggles to adapt on Level 3. Level 3 forces longer paths, bottlenecks, and role specialization, which require larger representational updates. EWC's cumulative constraints over-tighten the trunk and slow adaptation, while Online EWC's decay frees capacity for those shifts, so it learns the hard tasks more effectively. The multiple output heads alone are not enough. They isolate outputs, but the penalty sits on the shared backbone. When the trunk needs to be to rewired for new Level 3 tasks, EWC resists too much, while Online EWC allows it more. Moreover, credit assignment is noisier on Level 3 due to sparser effective signals and longer horizons. A single, stale Fisher snapshot can misdirect EWC's penalty. The rolling estimate in Online EWC smooths that noise and tracks the current regime more closely.

D Partially Observable MEALs

To more closely mimic the constraints faced by real-world agents, we introduce a *direction-aware* egocentric observation setting. Each agent perceives a rectangular window centered on itself, with tiles outside this window masked. The window is anisotropic with respect to the agent's heading: we separate forward, side, and rear extents, which increase with difficulty (Table 7). This scaling is intentionally balanced with the overall environment design: as the grid size grows with difficulty, the perceptual window also expands to maintain a comparable challenge-to-information ratio. Consequently, the tasks become POMDPs, where exploration, memory (e.g., recurrent state), and

Table 7: Field-of-view specification for the partially observable MEAL variant. Window size and directional extents scale with difficulty.

Difficulty	Grid Size	Forward View	Side View	Rear View	Obs Window (H×W)
Easy	6–7	1	1	0	2×3
Medium	8–9	2	1	0	3×3
Hard	10–11	3	2	1	3×5



(a) **Level 1**: 2×3 window. (b) **Level 2**: 3×3 window. (c) **Level 3**: 3×5 window.

Figure 10: Egocentric observation windows by difficulty. Visibility grows with difficulty but remains partial, preserving the need for exploration and memory.

implicit/explicit coordination provide tangible benefits. In particular, Level 1 removes rear context entirely, Level 2 extends the look-ahead by one tile, and Level 3 adds both longer look-ahead and rear visibility, reducing blind spots while preserving partial observability (Figure 10).

E Curriculum Learning

In all training settings, agents consistently struggle on Level 3 tasks with large grids. Curriculum learning has been shown to improve final performance on difficult tasks by gradually increasing task complexity [5, 28, 31]. We investigate whether a simple difficulty-based curriculum can help agents better learn harder MEAL tasks under the same data budget. To this end, we design a curriculum sequence where each difficulty level contributes an equal number of tasks. Specifically, we sample 5 layouts each from Level 1 (easy), Level 2 (medium), and Level 3 (hard), and present them in ascending order of difficulty (layouts 1–5, then 6–10, then 11–15). As a baseline, we compare with a default sequence that trains on 15 hard (Level 3) layouts without any prior exposure to easier tasks. Performance is evaluated based on the normalized average score over the 5 tasks in the sequence of the respective difficulty.

Table 8: Curriculum vs. default training under an equal data budget. We report the average score over the task windows of the respective difficulty.

Strategy	Medium (6–10)	Hard (11–15)
Default	0.693 ± 0.147	0.328 ± 0.238
Curriculum	0.668 ± 0.152	$\mathbf{0.653} \pm \mathbf{0.181}$

The results in Table 8 show no statistically significant difference between the two strategies on Level 2, given the high variance. However, on Level 3, the curriculum strategy nearly doubles performance. A plausible explanation is that, under curriculum training, the agent first experiences 5 easy and 5 medium tasks, where it receives denser reward signals and more frequent successes. This exposure likely builds useful priors and stabilizes learning, improving adaptation to harder tasks later. In contrast, the default strategy trains only on hard tasks throughout the sequence, where exploration is more challenging and initial rewards are more difficult to obtain, leading to weaker performance overall.

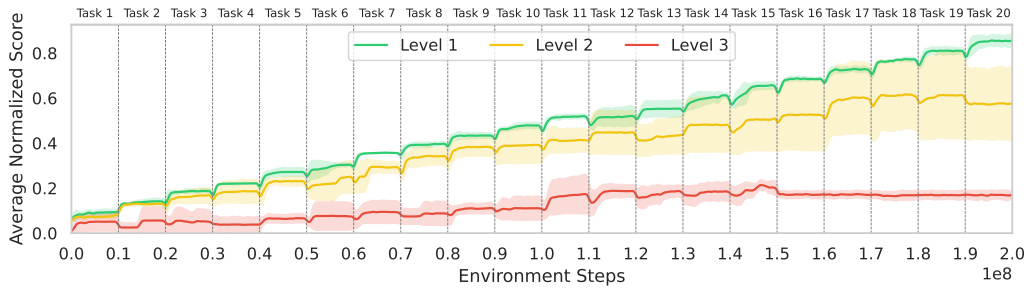


Figure 11: **Average Normalized Score** over the course of training EWC on a sequence of 20 randomly generated tasks per difficulty level. Shaded regions indicate 95% confidence intervals across 5 seeds.

F Difficulty Levels

Higher difficulty levels pose greater challenges for both learning and retention. As the grid size and obstacle density increase, the environment becomes more complex: interactable items are farther apart, and navigation paths are longer and more convoluted. This increases the number of steps required to complete a recipe, making the overall task harder to memorize and reproduce. Higher-level layouts also add demands for plasticity and transfer. The larger layout space introduces greater variability between tasks, making it harder to reuse learned behavior. These factors collectively lead to lower performance as difficulty increases, as shown in Figure 11. EWC performs reliably on Level 1, successfully learning new tasks while retaining performance on learned ones. On Level 2, learning remains effective up to around the midpoint of the sequence, after which progress slows and variance across runs increases markedly. In Level 3, EWC struggles from the outset, as performance is poor even on early tasks, and learning new tasks stalls entirely after the 15th task in the sequence.

G Network Plasticity

G.1 Metrics

We follow Abbas et al. [1], Dohare et al. [12] and quantify **plasticity**, the ability to fit fresh data after many tasks, by three complementary metrics computed from the training reward.

Notation. For a single task let r_t be the online reward at step $t \leq T$. A repetition experiment presents the same task R times, so the trace splits into R contiguous segments of equal length $L = T/R$. We smooth r_t with a Gaussian kernel (bandwidth σ) and define the cumulative average

$$\bar{r}(t) = \frac{1}{t}\sum_{i=1}^{t} r_i, \qquad t = 1, \ldots, L.$$

All metrics compare a later repetition $j > 0$ with the *baseline* repetition $j = 0$.

AUC–loss. Let $\mathrm{AUC}_j = \int_0^L \bar{r}_j(t)\, dt$. The capacity drop for repetition j is

$$\mathrm{loss}_j \;=\; 1 - \frac{\mathrm{AUC}_j}{\mathrm{AUC}_0}, \qquad j = 1, \ldots, R-1, \tag{5}$$

where 0 indicates perfect retention. We report the mean of Eq. (5) over repetitions and seeds.

Final-Performance Ratio (FPR). With $p_j = \bar{r}_j(L-1)$ the plateau reward of repetition j,

$$\mathrm{FPR}_j \;=\; \frac{p_j}{p_0}, \qquad j = 1, \ldots, R-1, \tag{6}$$

so $\mathrm{FPR}_j > 1$ implies no loss, $\mathrm{FPR}_j < 1$ indicates degraded plateau performance.

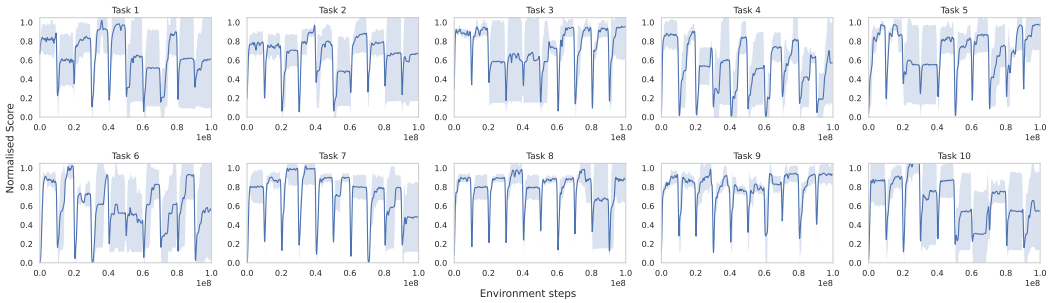


Figure 12: Training curves of FT across a Level 1 10-task sequence repeated ten times over 5 seeds.

Table 9: Homogeneous vs. heterogeneous (designated roles) 2-agent training results over Level 1 20-task generated sequences using shared rewards and IPPO in combination with EWC.

Setting	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
Homogeneous	$\mathbf{0.90 \pm 0.04}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.20 \pm 0.08}$
Heterogeneous	0.68 ± 0.09	0.03 ± 0.02	-0.05 ± 0.09

Raw-AUC Ratio (RAUC). Using the *unsmoothed* running reward,

$$\mathrm{RAUC}_j \;=\; \frac{\mathrm{AUC}_j^{\mathrm{raw}}}{\mathrm{AUC}_0^{\mathrm{raw}}}, \qquad j = 1, \ldots, R-1, \tag{7}$$

which captures the total reward accumulated during learning. Higher values in Eq. (6)–Eq. (7) are better.

Sequence-level aggregation. For a task sequence of length $|\mathcal{T}|$ we compute the per-task means of (5)–(7) and average across tasks, yielding a single global score per repetition count R.

G.2 Training Curves

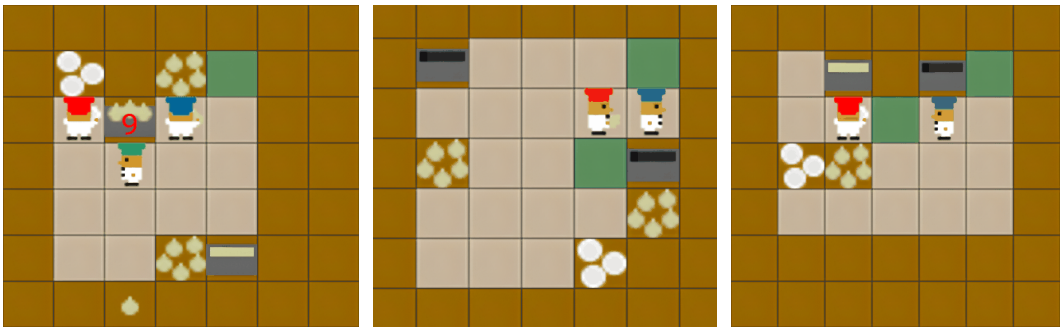
Figure 12 plots the mean normalized score of the fine-tuning (FT) baseline over ten repetitions. Performance on Tasks 8 and 9 remains virtually unchanged, indicating little to no plasticity loss. In contrast, Tasks 1, 2, 6, and 10 show a clear degradation: the agent fails to recover the score achieved during the first repetition, illustrating a pronounced loss of plasticity.

H Designated Roles

In Overcooked, agents are identical in their capabilities and attributes. However, in many real-world scenarios, autonomous agents either 1) possess different physical properties or 2) are functionally identical but are expected to fulfill distinct, complementary roles to cooperate effectively for a common goal. To capture this dimension in MEAL, we design a heterogeneous agent setting with **designated roles**.

In this variant, two agents are randomly assigned one of the two predefined roles at the start of each task: **chef** and **waiter**. The chef is responsible for preparing the soup by loading onions into the pot, but cannot pick up plates. The waiter handles dish delivery but cannot pick up onions. This enforces complementary capabilities, meaning neither agent can complete the full recipe alone, meaning that successful catering requires coordinated role execution and adaptation. Note that the roles are sampled per task and may switch across tasks, making continual learning essential.

We evaluate this setting over 20-task Level 1 sequences using EWC with IPPO under shared rewards. Table 9 compares the heterogeneous setup to the default homogeneous setting. We observe a clear performance drop in the role-restricted setting, as throughput decreases when agents are limited to certain actions and cannot flexibly switch between tasks. Another factor is asymmetric step costs: in many layouts, loading the pot with 3 onions takes more steps than a single plate-and-deliver trip,



(a) **Single-pot fixation**. All agents are clustered around a single pot, waiting for it to finish cooking, while ignoring a ready soup in the bottom pot.

(b) **Deadlock**. The red agent tries to place an onion into the pot, but is blocked by the blue agent, who cannot move aside.

(c) **Role collapse.** One agent completes the pipeline solo while the other wanders or idles. The policy settles on a local minimum.

Figure 13: Qualitative failure modes observed in Overcooked. All behaviors stem from inadequate coordination, limited exploration, or insufficient role allocation.

making the chef the throughput bottleneck. Generalization also suffers as agents struggle to transfer knowledge when their roles change across tasks, since skills learned in one role do not apply to the other. This role-switching dynamic further exacerbates forward transfer challenges in continual learning.

I Common Pitfalls

Despite shared rewards and simple layouts, learned policies frequently fall into recurring failure modes that throttle throughput and coordination. Figure 13 illustrates three such patterns we observe consistently across layouts and levels.

J Extended Results

In this section, we provide additional experimental results. Tables 10, 11, 12 add 95% confidence intervals to the main baseline results. Figure 16 depicts the per-task evaluation curves of Level 1. Figure illustrates forward transfer.

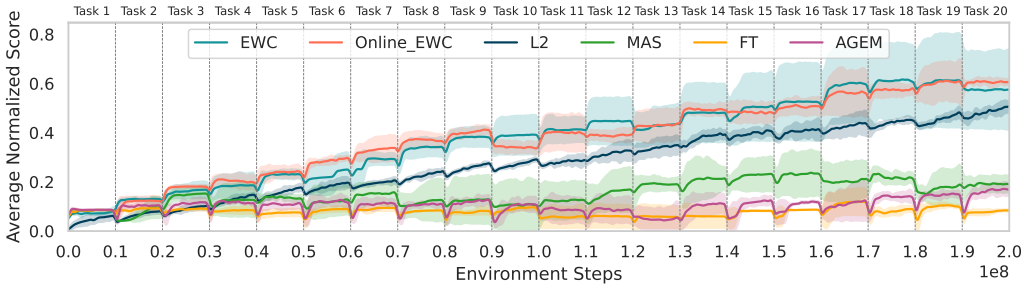


Figure 14: **Average Normalized Score** curves on Level 2.

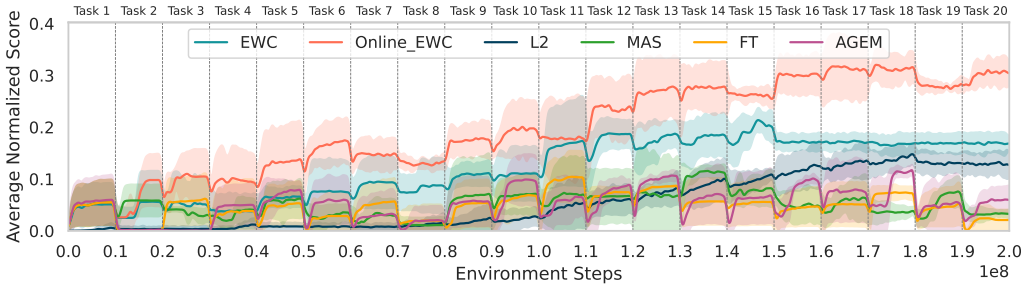


Figure 15: **Average Normalized Score** curves on Level 3.

Table 10: Level 1 baseline results with confidence intervals.

Method	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
FT	0.048$\pm$0.00	0.899$\pm$0.01	0.201$\pm$0.03
EWC	**0.839**$\pm$0.03	0.031$\pm$0.03	0.055$\pm$0.06
Online EWC	0.769$\pm$0.09	0.150$\pm$0.09	**0.208**$\pm$0.03
MAS	0.281$\pm$0.07	0.501$\pm$0.06	-0.233$\pm$0.03
L2	0.753$\pm$0.02	**0.031**$\pm$0.00	-0.199$\pm$0.09
AGEM	0.204$\pm$0.05	0.761$\pm$0.07	0.125$\pm$0.10

Table 11: Level 2 baseline results with confidence intervals.

Method	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
FT	0.041$\pm$0.01	0.790$\pm$0.01	0.065$\pm$0.02
EWC	**0.604**$\pm$0.21	**0.061**$\pm$0.01	-0.086$\pm$0.34
Online EWC	0.585$\pm$0.03	0.211$\pm$0.03	**0.152**$\pm$0.05
MAS	0.155$\pm$0.09	0.423$\pm$0.07	-0.355$\pm$0.06
L2	0.496$\pm$0.02	0.106$\pm$0.01	-0.527$\pm$0.04
AGEM	0.117$\pm$0.01	0.625$\pm$0.07	-0.083$\pm$0.07

Table 12: Level 3 baseline results with confidence intervals.

Method	$\mathcal{A}\uparrow$	$\mathcal{F}\downarrow$	$\mathcal{FT}\uparrow$
FT	0.010$\pm$0.02	0.519$\pm$0.08	-0.157$\pm$0.20
EWC	0.178$\pm$0.02	0.053$\pm$0.06	-0.650$\pm$0.13
Online EWC	**0.306**$\pm$0.00	0.168$\pm$0.02	**-0.149**$\pm$0.15
MAS	0.034$\pm$0.01	0.245$\pm$0.07	-0.542$\pm$0.20
L2	0.127$\pm$0.03	**0.052**$\pm$0.00	-0.827$\pm$0.04
AGEM	0.037$\pm$0.02	0.457$\pm$0.05	-0.169$\pm$0.18

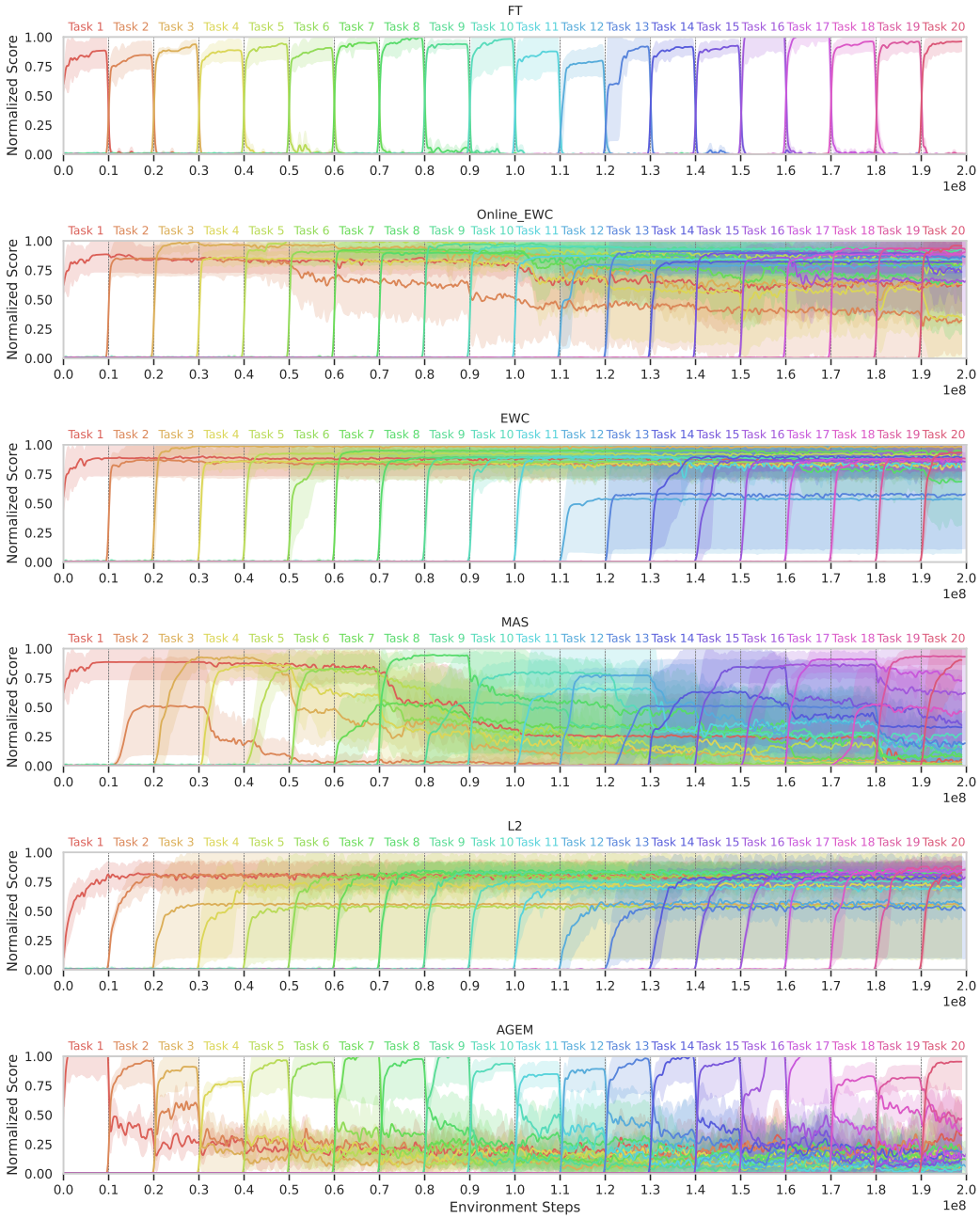


Figure 16: The **evaluation curves** of Level 1 illustrate the extent of forgetting across tasks. FT suffers from clear catastrophic forgetting: once the agent transitions to a new task, performance on the previous task collapses immediately. EWC and L2 display near-perfect retention.

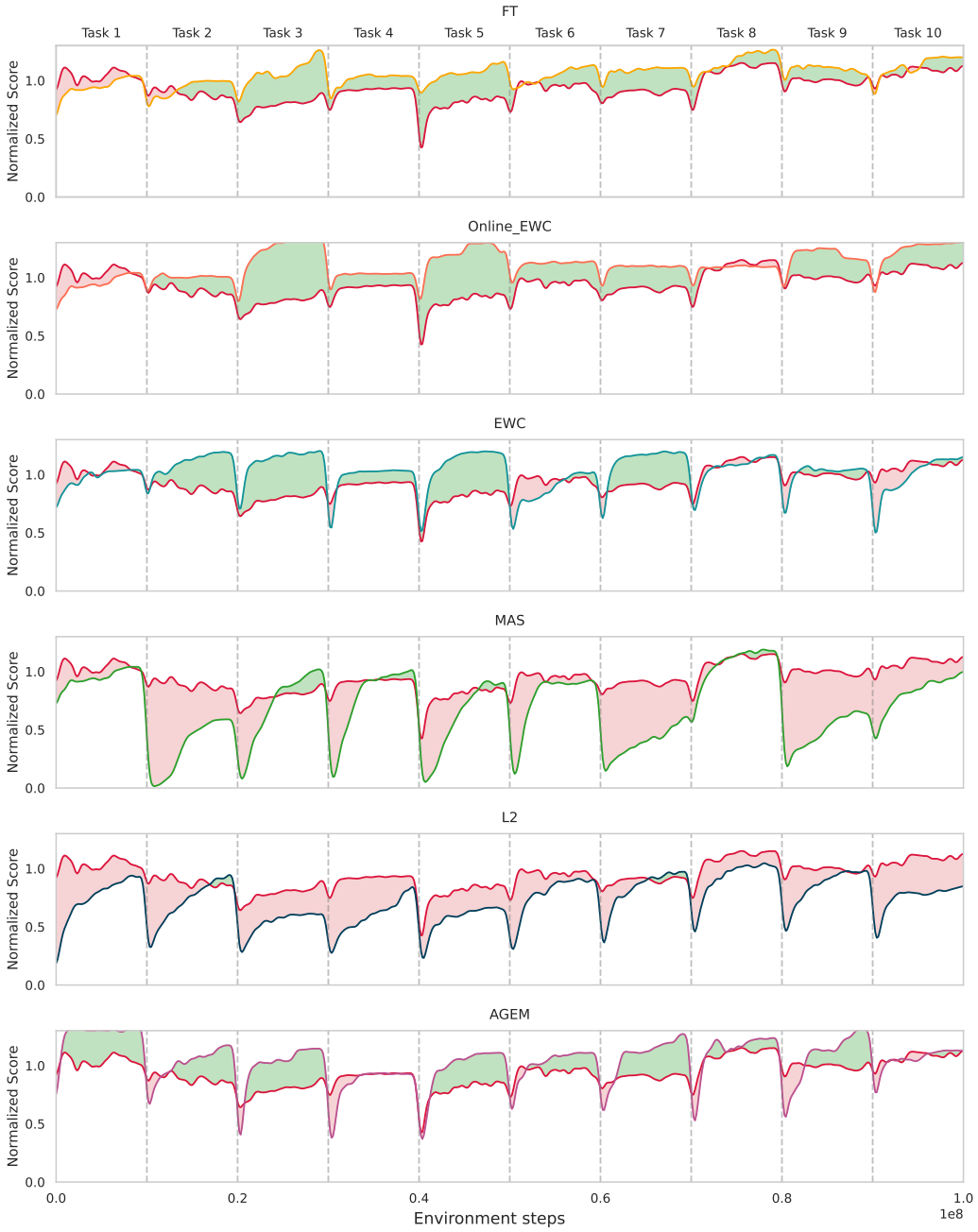


Figure 17: **Forward transfer** on Level 1. The green shaded areas depict positive transfer compared to the IPPO baseline, and the red shaded areas show negative transfer.