Combating Catastrophic Forgetting and Interference in Continual Reinforcement Learning

Adam Gans, Elijah Tabachnik, Juan Martinez, Harshvardhan Sharaff

University of California, Irvine — Irvine, CA, USA
{*agans1, etabachn, juanrm3, hsharaff*}*@uci.edu*

Abstract—Reinforcement Learning (RL) agents are sometimes asked to do multiple different tasks consecutively. For these agents, this is a difficult proposition, as they have trained for a specific defined task and now must rewire their thinking to a different task. This challenge is well known in reinforcement learning research and has a name: catastrophic forgetting. We attempt to study this challenge in a MiniGrid environment where the reward for the regime switches periodically between two goal configurations, which simulates a different task having to be learned by the agent. The vast majority of research on continual learning gives the agent some sort of input or signal to inform it which task it should accomplish, but we diverge from this by forcing our agents to have to learn on their own when the current task changes, which is made especially difficult as we use tasks that are contradictory. This leads us to the other problem we will face; catastrophic interference, where the tasks we want to learn directly interfer with each other. We compare two baselines and three distinct strategies for combating catastrophic forgetting. First, we establish a standard Proximal Policy Optimization (PPO) Deep RL baseline. Then we created a Dyna-PPO baseline that improves the PPO agent with a lightweight CNN world model that learns to predict the world and guides training with imagined roll outs that provide additional policy updates. For our first attempted strategy, we introduce a regime-specific Mixture of World Models (MoWM) architecture that dynamically spawns and routes through specialized world models based on surprise, enabling automatic regime detection and per-regime world models, allowing a single PPO agent to train on all world model dreams at once, learning every regime simultaneously. In our second strategy, we evaluate a double RL approach in which an outer ”brain” agent learns to adjust the hyperparameters of an inner policy-learning agent, ultimately trying to optimize the inner agent’s reward and recovery across regime switches. For our third strategy, we introduce a transformer-based world model built to handle and learn multiple regimes at once, enabled by a lightweight saved memory buffer, that both predicts the next world state/reward and provides the PPO agent a context vector, allowing it to distinguish which regime it’s in and learn multiple regimes at once. We evaluate each strategy on a Dual-Goal MiniGrid benchmark, measuring recovery speed after regime switches and retained performance upon regime recurrence. Our results demonstrate the trade-offs between combined vs. separate networks, imagined outputs, and dedicated regime-specific specialization for catastrophic forgetting and interference in continual RL.

1 Introduction

Deep reinforcement learning (RL) agents typically assume a stationary environment, where goals are set in stone, and the environment rules are static. However, in the real world the environment is inherently non-stationary, meaning reward functions, transition dynamics, and goals can shift over time. A well-documented problem under such conditions is catastrophic forgetting, where adapting to a new task degrades competence in previously mastered tasks. However, the majority of research on continual learning gives the agent some sort of input or signal to inform it which task it should accomplish, but we diverge from this by forcing our agents to have to learn on their own when the current task changes, which is made especially difficult as we use tasks that are directly contradictory, leading models to catastrophic interference. We study this problem in a controlled testbed: a custom MiniGrid Dual-Goal environment in which a "desired" goal switches periodically, forcing the agent to repeatedly relearn and retain competing reward/physics mappings, without any signal or visual cue that the regime changed. We implement and compare five agent architectures, two standard baselines, and three novel strategies:

1) A standard PPO baseline.
2) A simple world model, Dyna-PPO baseline [Schulman et al.(2017)].
3) A regime-specific Mixture of World Models that dynamically spawns and routes specialized world models based on surprise while using all of the world models to "dream" train the single PPO agent.
4) A meta-learning agent in which an outer "brain" RL agent adjusts the hyperparameters of an inner policy-learning agent to optimize long-term performance across task switches.
5) An advanced Transformer world model that predicts the next state/reward based on the previous 30 steps played, enabled by a lightweight memory bank, allowing it to learn multiple regimes at once and communicate these to the PPO agent.

Our central questions are (1) how severely does a standard PPO agent forget under repeated task switches? (2) Can a surprise-routed MoWM automatically detect task shifts and maintain per-task learning reliably? (3) Does hierarchical control through our double RL approach improve adaptation across tasks? (4) Can a single Transformer world model learn multiple regimes simultaneously given the recent game history?

2 Data

Our data consist of agent-environment interaction statistics collected during training.

2.1 Environment: Dual-Goal MiniGrid

We use a custom 8×8 MiniGrid environment [Chevalier-Boisvert et al.(2018)] containing two colored goal tiles (e.g., green and blue). The agent starts at a random position and faces a random direction within a walled grid. The agent then goes through an episode that is an attempt by the agent to reach the goal. Each episode ends when the agent reaches a goal or exceeds a 256 timestep limit. A full training run consists of thousands of these episodes. At any given time, exactly one of the two goals is designated as correct, while the other is a failure. Reaching the correct goal gives a reward of $+5$, reaching the incorrect goal yields -1, and a per-step penalty of -0.01 is applied at every timestep.

2.2 Dynamic Rule Changes via Regime Switching

To emulate dynamic rule changes and cause catastrophic forgetting, we periodically switch which goal color is correct. We define a *regime* as a set segment of training in which one goal is correct. After a fixed number of environment steps, the roles of the two goals are swapped:

- During regime 0, the green goal yields $+5$ and the blue goal yields -1.
- During regime 1, the green goal yields -1 and the blue goal yields $+5$.

From the agent's perspective, none of the inputs change. Thus, it can only realize that the regime shifts after an episode ends and it received a different reward than it expected. We consider repeated regime switches over the full course of training, experimenting with switch intervals of 100,000 steps (fast) and 296,000 steps (slow). Total training ranges from 300K to 2.5M environment steps, depending on the experiment. We used much higher step counts for the transformer, as it takes much longer to train, as we will talk about later.

2.3 Observations and Action Space

Observations are represented as a one-hot tensor with C channels over the grid. In our implementation, $C = 21$, corresponding to MiniGrid's encoding of walls, floor, agent, and goal objects, giving an observation shape of $(C, H, W) = (21, 8, 8)$. This input is suitable to be used by CNN encoders and to represent discrete next-state prediction in the world model. The action space is reduced to three discrete actions: turn left, turn right, and move forward. We utilize MiniGrid's already existing transition mechanisms for movement and collisions with walls. Training is parallelized across 16 environment instances to accelerate model training and data collection.

3 Methods: Architecture Design and Training

3.1 Baseline 1: Proximal Policy Optimization

Our first baseline agent is Proximal Policy Optimization (PPO) [Schulman et al.(2017)] implemented with an actor-critic architecture, an actor to predict and a critic to judge this prediction, which uses generalized advantage estimation (GAE) as a signal to use to judge whether a prediction was good or not. The actor and critic functions share a three-layer convolutional encoder ($32 \to 64 \to 64$ channels, 3×3 kernels with padding, ReLU activations) followed by separate two-layer MLP heads that calculate policy scores

and estimated future rewards. Given a batch of on-policy paths to take, we compute advantages using GAE and update the actor function with a clipped loss function while at the same time training the critic to calculate estimated future rewards more accurately and a random bonus term to encourage exploration and to prevent overfitting. Training is parallelized across 16 synchronous environments.

3.2 Baseline 2: Simple World Model

The Dyna-PPO agent expands on the PPO backbone with a lightweight CNN world model f_ϕ that predicts both the next observation and immediate reward from the current observation and action. Formally,

$$(\hat{o}_{t+1}, \hat{r}_t) = f_\phi(o_t, a_t),$$

where $o_t \in R^{C\times H\times W}$ is a one-hot grid observation, a_t is a discrete action, $\hat{o}_{t+1}$ is the next-state prediction, and $\hat{r}_t$ is a scalar reward prediction. The world model consists of: (i) a convolutional encoder that matches with the actor-critic encoder, (ii) an action embedding combined with the encoded observation, and (iii) two output heads, one for next-state prediction and one for reward. The world model learns from actual experiences that the agent collects while training and uses these experiences and the loss they give through cross-entropy loss and mean squared error to learn and estimate better parameters.

3.3 Intrinsic Curiosity from Prediction Error

We derive an intrinsic reward, or the reward that the agent gives itself, from the world model's prediction error to attempt to ascribe to curiosity-driven exploration [Pathak et al.(2017)]. For each transition, we compute

$$\mathcal{L}_{\text{state}} = \text{CE}(\hat{o}_{t+1}, o_{t+1}), \quad \mathcal{L}_{\text{reward}} = \text{MSE}(\hat{r}_t, r_t),$$

where $\mathcal{L}_{\text{state}}$ is how wrong the world model was about the next state, or our cross-entropy loss, and $\mathcal{L}_{\text{reward}}$ how wrong the world model was about the reward, or our mean squared error. These combine to make our surprise metric, or how wrong the prediction was from the reality. We define the intrinsic reward as

$$r_t^{\text{int}} = clip\big(\beta \cdot (\mathcal{L}_{\text{state}} + \mathcal{L}_{\text{reward}}), 0, r_{\max}\big),$$

with scaling coefficient β and clip $r_{\max}$. The intrinsic reward is scaled by our total surprise with it β so that it doesn't overwhelm the external rewards given by the environment, as total surprise can be large numbers. It is clamped between 0 and $r_{\max}$ so that it isn't negative and it doesn't overwhelm the reward function when surprise spikes. The intrinsic reward makes sure that our agent is guided both by the environment itself and a small amount by surprise.

The total reward used for PPO updates is

$$r_t^{\text{tot}} = r_t^{\text{ext}} + r_t^{\text{int}}.$$

where r_t^{ext} is external reward given by the environment and r_t^{int} is the intrinsic reward that the agent rewards itself.

3.4 Dreaming: Imagined Rollouts for Policy Updates

Implementing Dyna [Sutton(1990)], we periodically generate imagined paths within the mini-grid by using the policy inside the world model for a short interval. We create the imagined paths by starting from real states the agent actually saw while playing. At each imagined starting point within these previous real states, the agent selects an action, and the world model imagines what would happen. The world model outputs a "fuzzy" predicted observation, which guesses an observation given an action. We force this "fuzzy" predicted observation back into a real observation with discretization to sharpen the prediction to how a real state would look, and prevent the accumulation of "blurry" imagined states that the PPO model cannot train on. Ultimately, we treat these imagined transitions as additional on-policy data to train our model: we compute scores on these policies using the critic function evaluated on dreamed states (with gradients frozen while dreaming) and perform extra PPO updates using the imagined trajectories.

3.5 Training Phases

Each training update consists of four phases:

1) **Real experience collection**: interact with 16 environments in parallel for a fixed number of steps, compute intrinsic rewards, and record real states that the agent encounters.
2) **Policy update on real data**: compute returns and advantages using GAE and run several epochs of PPO updates.
3) **World model update**: train the world model on the same real transitions with supervised losses.
4) **Dreaming (Dyna-PPO only)**: sample start states from real states that the agent previously encountered, imagine trajectories in the world model, and run one additional PPO epoch on dreamed data.

3.6 Regime-Specific World Model (MoWM)

To explicitly eliminate catastrophic forgetting in environments with discrete, non-stationary dynamics, we extend the Dyna-PPO framework with a Mixture of World Models (MoWM). Rather than forcing a single neural network to model conflicting physical laws, this architecture maintains a dynamic library of specialized, lightweight world models, each dedicated to a distinct environmental regime. Then the PPO agent trains on dreams from all of these world models, allowing it to learn all of them at once.

Base World Model Architecture: Each regime-specific model f_{ϕ_k} in the library predicts transition dynamics for a single regime k. Given a one-hot visual state s_t and discrete action a_t, the state is processed by a 3-layer CNN and concatenated with a learned action embedding. A 2-layer MLP then outputs next-state logits and a scalar reward prediction. To prevent compounding errors during generative rollouts, predicted next-state logits are discretized back into one-hot vectors via an argmax operation before recursive querying.

MoWM Manager: Surprise Detection and Routing: The MoWM acts as an orchestrator, actively tracking the sum of the state cross-entropy and reward MSE of the currently active world model. We maintain an Exponential

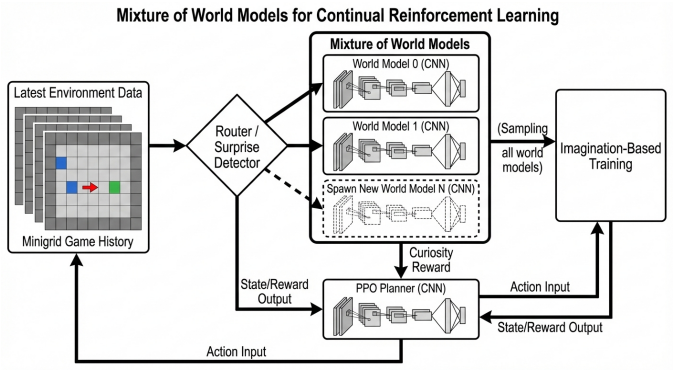


Fig. 1: MoWM Architecture Diagram.

Moving Average (EMA) of this prediction error. A regime switch is detected when the batch loss exceeds a dynamic threshold $L_{thresh} = \max(\text{EMA}, L_{floor}) \times \alpha$. Upon detecting a "surprise," the manager isolates the high-loss transitions and evaluates all inactive models within the library against this data. If an existing model successfully predicts the transitions (loss below a viability threshold), the manager routes the system to this veteran model. If no model succeeds, the manager spawns and initializes a new world model, assigning it a novel 'regime_id'.

Safe State Rollback and Policy Patience: Because RL agents collect data in batches, the active world model may inadvertently train on out-of-distribution data immediately following a regime switch. To prevent weight corruption, the MoWM maintains a rolling buffer of recent checkpoints and executes a "Safe State Rollback" upon surprise detection. Furthermore, a "patience" mechanism freezes PPO policy updates for $N_{patience}$ epochs, preventing the actor-critic from updating on off-policy shock data while the manager resolves routing ambiguities.

Generative Replay and Context-Aware Policy: The PPO agent is formulated as a context-aware policy $\pi_\theta(a_t|s_t, \text{regime\_id})$, parameterized by an Actor-Critic network that takes the currently routed world model's 'regime_id' as an explicit categorical input. To prevent the policy from forgetting older tasks, the MoWM manager samples starting states from a historical State Reservoir and queries the *inactive* world models to generate "dreamt" trajectories of past regimes. The PPO agent performs updates on a mixed batch of real transitions (from the active regime) and dreamed transitions (from historical regimes), allowing it to remember past regimes while learning the current one.

3.7 Meta-Learning and Neuromodulation

Our second main approach is a meta-learning system that wraps a full inner Dyna-PPO training run inside an outer Gymnasium environment (`MetaEnv`). Every outer step executes K inner PPO updates, where K is the *decision interval*, and returns a normalized 19-dimensional summary of the inner learner's state. The Brain observation includes performance signals (mean episodic return, success rate, and failure rate), surprise and world-model diagnostics (mean surprise, surprise delta, state loss, and reward loss), policy statistics (entropy, policy loss, and value loss), the current values of the controllable inner hyperparameters, the time since the most recent surprise spike, and episodic-memory utilization.

The outer Brain policy is itself trained with PPO. It uses a two-layer MLP actor-critic and a tanh-squashed Gaussian policy over a 15-dimensional continuous action space. Seven action dimensions control scalar inner-loop levers: learning rate, entropy coefficient, intrinsic curiosity coefficient, imagined rollout horizon, replay ratio, replay prioritization, and policy anchoring weight. The remaining eight dimensions form a context code for neuromodulation.

Neuromodulation in our framework gives the outer Brain a way to influence not only how the inner agent is trained, but also how it internally processes observations at inference time. In addition to selecting scalar hyperparameters such as learning rate, entropy coefficient, replay ratio, and anchoring weight, the Brain outputs an 8-dimensional context code that acts as a learned control signal. This code is passed to a context decoder inside the inner CNN actor-critic, which converts it into a feature-wise mask over the shared convolutional representation. Intuitively, this lets the Brain tell the inner agent which internal features should be emphasized or suppressed under the current regime, allowing the same policy network to behave differently across changing task conditions without requiring a separate model for each regime.

Formally, if h_t is the flattened shared CNN feature vector and $c_t \in [-1, 1]^8$ is the Brain's context code, the decoder produces a mask $m_t \in (0, 1]^d$ and the gated representation becomes $\tilde{h}_t = h_t \odot m_t$. The actor and critic then consume $\tilde{h}_t$ instead of the original features, so neuromodulation can change both the action distribution and the value estimate. The magnitude of this effect is controlled by the norm of the context code: when $c_t = 0$, the mask is all ones and the network reduces to its unmodulated baseline, while larger codes induce stronger feature suppression. This design makes neuromodulation a learned, context-dependent routing mechanism that can rapidly reconfigure the inner agent after regime switches, giving the Brain a direct way to shape behavior beyond ordinary hyperparameter control.

We log both the Brain's selected hyperparameters and dedicated neuromodulation diagnostics, including mask mean and standard deviation, per-channel mask activity, policy KL divergence relative to the unmasked policy, entropy change, and value shift. These diagnostics let us later test whether the context code behaves like a regime-sensitive routing signal rather than uncontrolled noise.

The implementation also supports a small number of imitation-style pretraining episodes before outer-loop PPO. During this warm start, a hand-coded heuristic increases exploration-oriented levers immediately after low success rates or surprise spikes, then tapers toward more exploitative settings as the inner learner recovers. During full Brain training, the outer reward can be configured as an area-under-the-curve score, a recovery-oriented shaping function, or a curriculum-style reward that up-weights successful revisits of earlier regimes.

3.8 Context-Aware Transformer World Model

A primary downside of the Mixture of World Models approach is that it needs to create and train a new model for

every different regime. To try and fix this, we introduce a Context-Aware Transformer World Model architecture. This system combines regime identification and world model prediction, utilizing a causal Transformer to infer hidden regime states and predict the next state/reward based on that knowledge. It utilizes a lightweight memory buffer to continuously train on and remember past regimes. Then we combine this with the PPO agent by giving the agent the latent context vector, h_t, as an input, allowing it to know which regime it's in.

Transformer World Model: The world model is a causal Transformer that models the transition dynamics over a sliding window of recent history, $S = 30$. At each timestep, the visual state s_t (processed via a 3-layer CNN), discrete action a_{t-1}, scalar reward r_{t-1}, and terminal flag d_{t-1} are projected into a unified $d_{model} = 256$ dimensional space. These tokens are summed with a positional encoding and processed by a 2-layer Transformer Encoder with 8 attention heads. The output at the final sequence step yields the latent context vector, h_t. The model is trained via self-supervised learning to predict the next visual grid (pixel-wise Cross-Entropy) and the next reward (Categorical Cross-Entropy across 256 discrete bins). We apply a burn-in mask to the loss, ensuring the Transformer is only penalized after observing at least one terminal state within the sequence, preventing from having to learn from random transitions.

Context-Aware PPO Agent: The decision-making agent is a feed-forward PPO network, similar to the baseline models or what's used by the MoWM. It extracts features from the current visual state s_t , and concatenates these features with the context vector h_t.

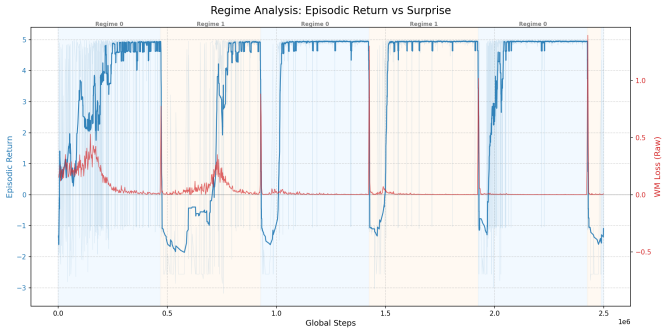
Memory Buffer and Generative Replay: To mitigate catastrophic forgetting of older regimes, we maintain a sequence memory buffer using reservoir sampling. Sequences are saved only if their prediction error (surprise) exceeds a moving threshold, calculated similarly to the MoWM EMA Threshold, and capacity is strictly allocated based on trajectory outcomes (60% failures, 20% successes, 20% neutral) to prevent class imbalance (storing more failures to balance out the success bias during live play). During policy updates, the Transformer simulates ("dreams") future transitions for a horizon $H = 5$ from seed states sampled from this buffer. The PPO agent performs updates on a mixed batch of live data and these generative replay trajectories.

3.9 Logging and Evaluation

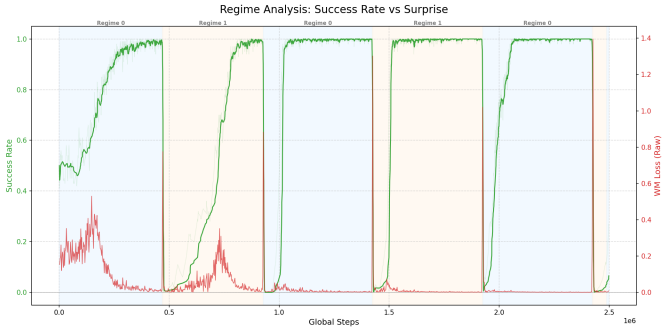
Training statistics are logged to TensorBoard. We record standard RL metrics (episodic return, episode length), success/failure/timeout rates, world model losses (state cross-entropy, reward MSE), and intrinsic reward statistics. Checkpoints are saved periodically to allow resumption and analysis during any part of the training process. For plots and deeper visual analysis, we post-process TensorBoard event files using a dedicated analysis script that reads logged statistics, applies smoothing, detects regime-switch boundaries, and exports visually informative plots.

TABLE 1: Key hyperparameters used in our experiments.

Parameter	Value
Total timesteps	300,000–2,500,000
Number of environments	16
Rollout length (steps)	128
PPO learning rate	3×10^{-4}
PPO update epochs / minibatch size	$4/256$
Discount factor γ	0.99
GAE parameter λ	0.95
Intrinsic coefficient β	0.015
Intrinsic clip $r_{\max}$	0.1
Imagined horizon H	10
World model learning rate	1×10^{-4}
Regime switch period (fast)	100,000 steps
Regime switch period (slow)	296,000 steps

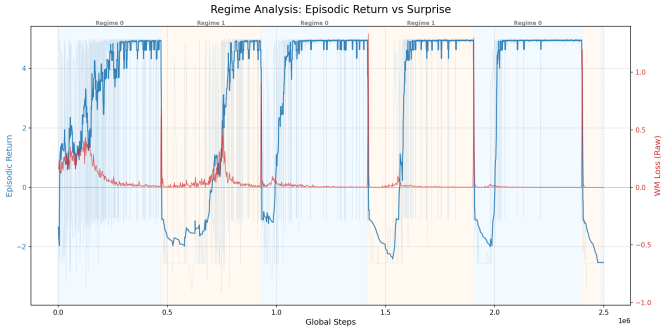


(a) Episodic return vs surprise

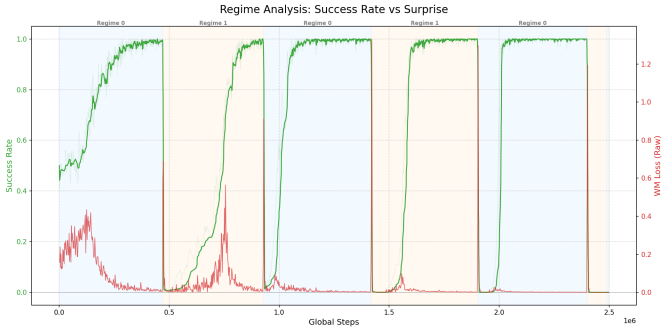


(b) Success rate vs surprise

Fig. 2: Learning using Standard PPO.



(a) Episodic return vs surprise



(b) Success rate vs surprise

Fig. 3: Learning using Dyna-PPO.

4 Results

We compare our five approaches, PPO baseline, Dyna-PPO Baseline, Regime-Specific MoWM, Meta-Learning, and the Context-Aware Transformer World Model. We are primarily looking at success rates, surprise/loss metrics, and how fast the agent converges/re-learns to quantify catastrophic forgetting and recovery.

4.1 Baseline Results

Figure 2 shows the evolution of episodic return and success rate over training of the standard PPO Baseline. Figure 3 shows the evolution of episodic return and success rate over training the Dyna-PPO Baseline. All agents eventually learn to reliably reach the correct goal; Dyna-PPO converges very slightly faster owing to its intrinsic curiosity bonus and additional "dream" training.

Under regime switching, performance drops sharply after each switch for both agents, but the degree of drop and speed of recovery vary slightly between the two methods.

4.2 Regime-Specific MoWM

The Regime-Specific MoWM explicitly trades memory footprint for perfect dynamics retention. As seen in Figure 4, the MoWM agent required only around 50,000 and 10,000 recovery steps after the regime switches.

Forgetting Mitigation via Generative Replay: When the environment reverted to previously seen regimes, the MoWM agent achieved convergence in an average of 30,000 steps before matching its performance from when the regime was first mastered. This demonstrates that continuous generative replay from the inactive world models successfully shields the context-aware PPO policy from catastrophic forgetting.

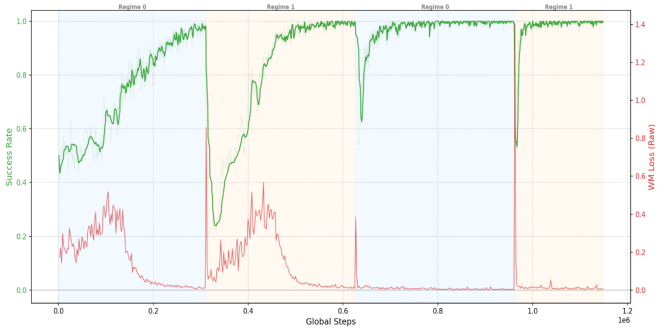
Autonomous Routing and Spawning Accuracy: The efficacy of the MoWM depends entirely on the manager's ability to detect shifts and route to the correct model. Viewing Figure 4b, you can see how the red surprise line only crosses the purple threshold line when the regimes truly shift, allowing the near-instantaneous shift of the PPO agent. Nearly the only reason that the success drops at all is due to the fact that the PPO must first fail for a full training batch before this world model routing happens.

4.3 Meta-Learning

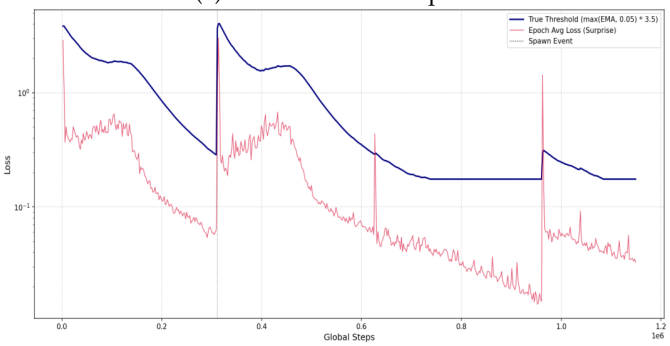
We study whether an outer-agent ("Brain") controller improves adaptation under regime switching by learning switch-contingent responses, rather than merely improving stationary training performance. The key question is whether the controller accelerates post-switch recovery once regime changes have been observed.

To stress this capability, we adopt a fast-switch schedule in which the inner agent is trained for $800{,}000$ total timesteps with regime switches every $100{,}000$ steps. This setting sharply exposes differences in recovery behavior.

Table 2 shows a clear hierarchy of performance across methods. The static baseline performs worst, achieving only 0.4638 average success and failing to reach 80% success in any regime. Performance collapses after each switch with



(a) Success rate vs surprise



(b) Surprise vs rolling surprise threshold

Fig. 4: Learning using MoWM Approach.

little evidence of recovery, indicating an inability to retain or rapidly reacquire useful behavior.

Introducing neuromodulation without learning (heuristic control) already yields substantial improvements. These configurations recover to high success levels in multiple regimes, demonstrating that selective plasticity alone can mitigate catastrophic forgetting by enabling faster readaptation after switches.

Brain-controlled scalar hyperparameter modulation further improves performance, achieving 0.7307 average success and consistently recovering across regimes. After the first regime, the agent reaches 80% success in $97{,}312$, $59{,}440$, and $51{,}168$ steps, and subsequently attains repeated 95% recoveries in later revisits. This indicates that learned meta-control over global training dynamics provides a stronger mechanism for continual adaptation than fixed or heuristic strategies.

Finally, combining Brain control with learned neuromodulation yields the strongest performance. The best configuration achieves 0.7498 average success and recovers in every regime, with multiple regimes reaching 95% success. This suggests that integrating global meta-control with structured, context-dependent modulation of the inner network produces the most effective adaptation under rapid switching.

Additional training curves illustrating these effects are provided in Appendix A (see Fig. 10 and Fig. 11).

Beyond aggregate performance, we analyze whether the neuromodulation context code encodes regime-specific information. Logged diagnostics—including mask statistics, per-channel activity, policy KL divergence relative to the unmodulated policy, entropy change, and value shift—allow

TABLE 2: Fast regime-switching experiments.

Experiment	Avg. Success Rate	Recovery Summary
Static baseline	0.4638	Failed to reach 80% success in all eight regimes. This run serves as the static-hyperparameter reference and shows poor recovery under repeated switching.
Heuristic neuromodulated run	0.6989	Failed to reach 80% in the first regime, then recovered to 80% in later regimes after 75,248, 45,424, 55,488, 89,648, and 33,104 steps, with two stronger recoveries reaching 95% after 95,140 and 76,768 steps.
Non-neuromodulated Brain run	0.7307	Failed to reach 80% in the first regime, then reached 80% after 97,312, 59,440, and 51,168 steps, followed by repeated 95% recoveries after 66,744, 71,848, 73,576, and 67,472 steps.
Neuromodulated Brain run	0.7498	Reached 80% in regimes 0, 1, 2, 4, and 6 after 78,560, 58,720, 33,552, 40,256, and 34,352 steps, and reached 95% in regimes 3, 5, and 7 after 52,304, 46,664, and 78,074 steps. This was the strongest of the four rapid-fire runs.

us to distinguish structured routing behavior from unstructured perturbations.

4.3.1 How Neuromodulation Works and How to Read the Diagnostic Dashboard

The neuromodulation dashboard(Figure 5) summarizes one Brain-controlled inner run at the exact timesteps when the Brain emits a new context code. The horizontal axis is therefore not raw episode index but *inner global step at Brain decision time*. Colored regime backgrounds and dashed vertical lines mark reward-regime switches. Read together, the four panels show the full causal chain from Brain output, to decoded mask, to changed network behavior.

The top panel ("Learning Progress and Regime Context") provides behavioral context for the rest of the dashboard. It overlays success rate, mean reward per step, and episodic return, so that drops in performance can be aligned with regime boundaries. When neuromodulation is useful, we expect these drops to be followed by recovery rather than prolonged collapse. This panel therefore tells us *when* adaptation is happening and whether changes in the lower panels occur near the moments when they matter most.

The second panel ("Brain Context Code") visualizes the 8-dimensional continuous context vector produced by the outer Brain. Each row is one context dimension and each column is one Brain decision. Red values indicate positive code entries and blue values indicate negative entries. If the Brain were not using this channel meaningfully, the heatmap would look like unstructured noise with no persistence. Instead, the code often forms temporally coherent bands and visibly changes near some regime boundaries, which is consistent with the Brain using the context code as a compact latent description of the current training situation.

The third panel ("Decoded Neuromodulation Mask") shows what happens after that 8-dimensional code is passed through the context decoder inside the inner actor-critic. The decoder maps the code to a feature-wise suppressive mask that is multiplied into the shared convolutional representation before both the actor and critic heads. In the dashboard, each row corresponds to one of the 64 shared CNN feature channels, and the plotted value is the mean mask strength for that channel after averaging over spatial positions. Values near 1 indicate channels that are largely preserved, whereas lower values indicate channels that are being suppressed. Vertical stripes in this heatmap therefore indicate moments when the Brain is reweighting which internal features the inner agent is allowed to rely on.

The fourth panel ("Neuromodulation Effect on Policy and Value") measures the downstream consequence of that masking. At each Brain decision, the same observation batch is run through the network twice: once with the current neuromodulation mask and once with an all-ones mask corresponding to "no neuromodulation." From those two forward passes we log three quantities: (i) the KL divergence between the masked and unmasked action distributions, (ii) the difference in policy entropy, and (iii) the mean absolute difference between the masked and unmasked value estimates. Formally, if π_m and π_u denote the masked and unmasked policies and V_m, V_u the corresponding value predictions, we track $\mathrm{KL}(\pi_m \,\|\, \pi_u)$, $H(\pi_m) - H(\pi_u)$, and $E[|V_m - V_u|]$.

This final panel is the clearest direct test of whether neuromodulation is actually doing anything. If the context code were being ignored, all three traces would remain at or extremely near zero. Instead, the policy-KL curve stays consistently above zero, the entropy-delta curve also remains nonzero, and the absolute value-delta curve exhibits the largest excursions. In other words, the mask changes both the actor and the critic, but in this run it affects the critic more strongly than the action distribution itself. That pattern is reasonable: a scalar value estimate is often more sensitive to feature suppression than the ranking of a small discrete action set. The graph therefore supports the interpretation that neuromodulation is not merely an auxiliary logging signal; it is actively changing the computation performed by the inner policy/value network.

One caveat is that the three traces in the bottom panel do not share the same natural units, even though they are plotted on a common axis for convenience. Their magnitudes should therefore be interpreted relative to their own zero baselines rather than directly compared numerically. Even with that caveat, persistent nonzero values in all three traces provide strong evidence that the Brain's context code is functioning as a genuine control signal for internal feature routing. More specifically, the Absolute Value Delta vs Unmasked trace shows pronounced transient changes immediately following regime switches suggesting that the outer-agent has learned to use neuromodulation to impact inner-agent learning.

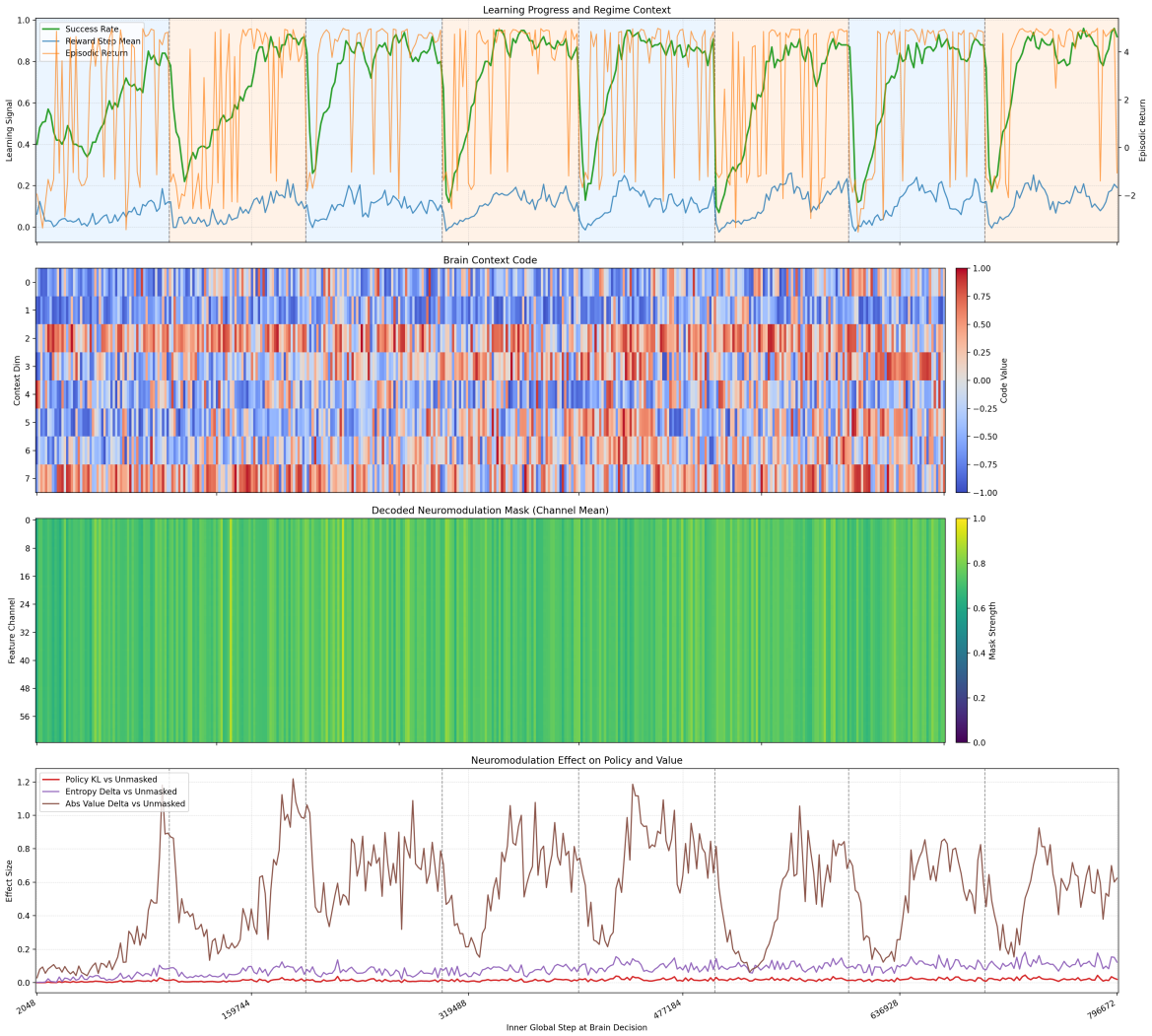


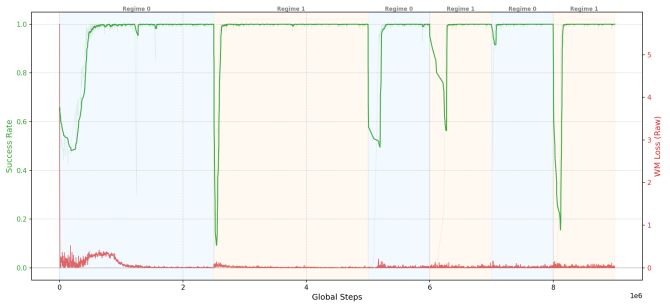
Fig. 5: Neuromodulation diagnostic dashboard for a Brain-controlled run. Each panel shows a stage in the causal chain from context code to behavioral effect at Brain decision timesteps. The horizontal axis corresponds to inner global steps at Brain decision points, with regime switches indicated by vertical markers.

4.4 Context-Aware Transformer World Model

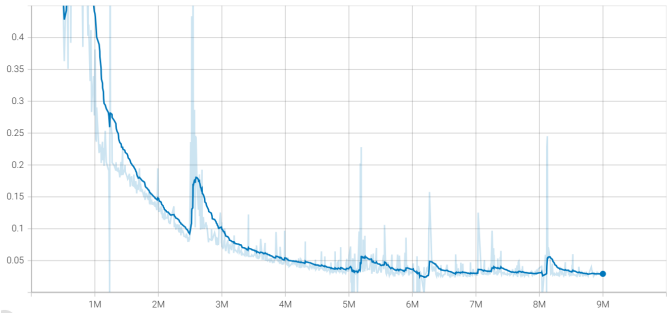
Finally, evaluate the Transformer World Model. The transformer world model was very difficult to create and fine-tune, and despite showing promises of working well, we simply ran out of time to build it to it's full potential. Our latest run results are shown in Figure 6. First, notice how many more steps we were required to run for the world model to achieve decent results in it's predictions. Because of this, we decided to switch regimes every 2.5 million time steps, then every 1 million after the 5 million steps mark.

World Model Loss: When returning to previously seen regimes, the Transformer world model has very small spikes, as you can see in Figure 6b, and it continues to drop in loss, showing that it is capable of learning multiple regimes simultaneously. Furthermore, we track the cosine similarity between two context vectors, created from sequences played the exact same way, but from two different regimes. This shows how well the world model can decipher different regimes. As you can see in Figure 6c, the cosine similarity drops all the way to 0.5, showing that the world model is very good at seperating the two regimes.

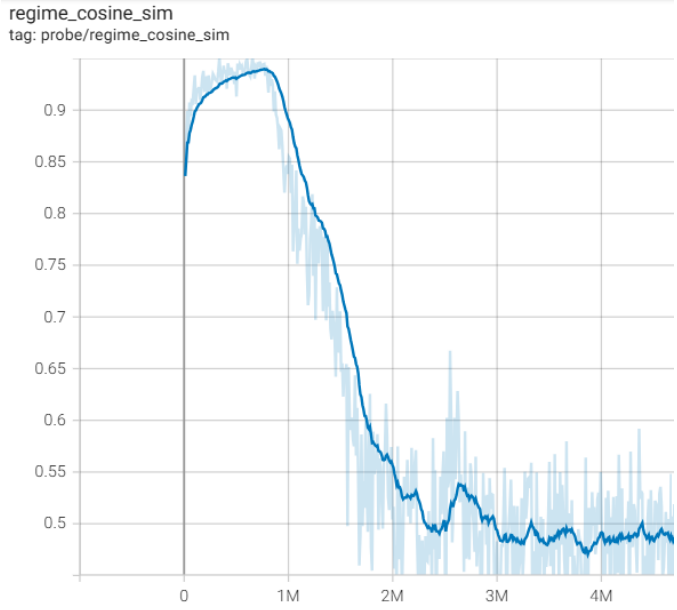
The Dreaming Problem: Despite this success by the world model, when we implemented the PPO dreaming, the PPO still suffered from catastrophic forgetting. This is the problem that we are actively trying to solve. One issue that contributes to this is that the latent context vector created by the transformer "drifts" throughout training, rather than remaining deterministic as the PPO requires. Another issue is that if the world model isn't 100% perfect, than it can lead the PPO astray in dangerous ways. These are some of the problems that lead to the catastrophic forgetting of this architecture to still exist.



(a) Success rate vs surprise



(b) World Model Loss (Zoomed In)



(c) Regime 0 vs Regime 1 Context Vector Cosine Similarity

Fig. 6: Learning using Transformer World Model approach.

5 Conclusion and Discussion

We studied catastrophic forgetting and interference in a non-stationary Dual-Goal MiniGrid environment and evaluated three strategies beyond just the standard PPO and Dyna-PPO approaches. Our results show that: (1) standard PPO agents and a Dyna-PPO agents learn the task in stationary conditions but forget drastically under repeated regime switches, relearning slowly; (2) a regime-specific Mixture of World Model with surprise-based routing and cross-regime dreaming mitigates forgetting by isolating per-regime dynamics; (3) a Meta-Learning approach, in which an outer agent alters an inner agent's hyperparameters, offers a complementary meta-learning perspective on continual adaptation; and (4) a Transformer World Model can learn multiple regimes at once with the help of a memory buffer, but further work will be needed to integrate this with a decision making (PPO) agent to complete it.

5.1 Other Approaches Attempted

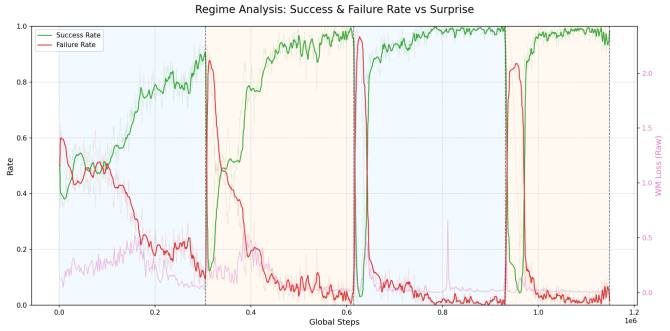
We also tried a multi-head actor-critic architecture designed to have our actor-critic agent have multiple heads, where each head was associated with a unique regime. The architecture would have unique per-regime policy decisions while also having a shared trunk that held common shared low-level visual features applicable to any regime. When a regime switch would be detected, per surprise spikes, the agent would switch to the appropriate regime, theoretically allowing it to immediately regain its previously learned policy while being fully isolated from other regimes. In addition to the multi-head architecture, another novel aspect of this approach was its use of Elastic Weight Consolidation [Kirkpatrick et al.(2017)], a normalization approach intended to curb catastrophic forgetting by discouraging large changes to encoder weights that were important for the previous regimes. This was used as an attempt to prevent important weights from being erased. EWC would only be applied to the grid encoder, as the per-regime heads by design couldn't be overridden by other regimes. This approach attempted to combine isolated networks with regularization on shared networks representation to solve forgetting. However, it also added more complexity. Exploratory experiments show 7 initial results not improving too much or, in fact, not improving at all from baseline.

5.2 Limitations

Several limitations remain. First, our evaluation is restricted to a single MiniGrid Gridworld benchmark with only 2 regimes; while this is good to test out approaches to solving catastrophic forgetting, the mechanisms are simpler than many real-world tasks would be. Second, most of these approaches introduced new neural networks or other strains on memory and/or time complexity, which limits the useful scenarios these architectures can apply. Third, the Meta-Learning approach introduces additional complexity in defining the outer agent's state and reward, and its benefits depend on the granularity of hyperparameter control. Its benefit is also questionable when the environment is simple, gaining performance when the environment is more complex.

5.3 Future Work

Future directions include: (1) extending to larger observation spaces and more complex tasks, such as continuous-control benchmarks or partially observable environments; (2) investigating alternative intrinsic motivation signals less reliant on prediction error, such as information gain or disagreement-based exploration; (3) continuing research on how to implement a decision making agent to learn from the transformer world model predictions; and (4) combining the strengths of the three approaches. For example, using the meta-learning outer agent to dynamically select between



(a) Success rate vs surprise over time.

(b) EWC Loss Dynamics (Shared Trunk)

Fig. 7: Multi-head architecture results. Left: Success rate (green) and world model reward-prediction loss (red, right axis) across regime switches. Right: EWC normalization penalty (red) and normal policy loss (blue). What is important to note is that both types of penalties are on about the same level in the graph, demonstrating that the loss is balanced between adapting to new observations (policy loss) and remembering important past weights (EWC penalty).

single-head and multi-head world model configurations based on detected non-stationarity.

APPENDIX A
APPENDIX: BRAIN EXPERIMENTS

A.1 Static Baseline vs Non-Neuromodulation Brain

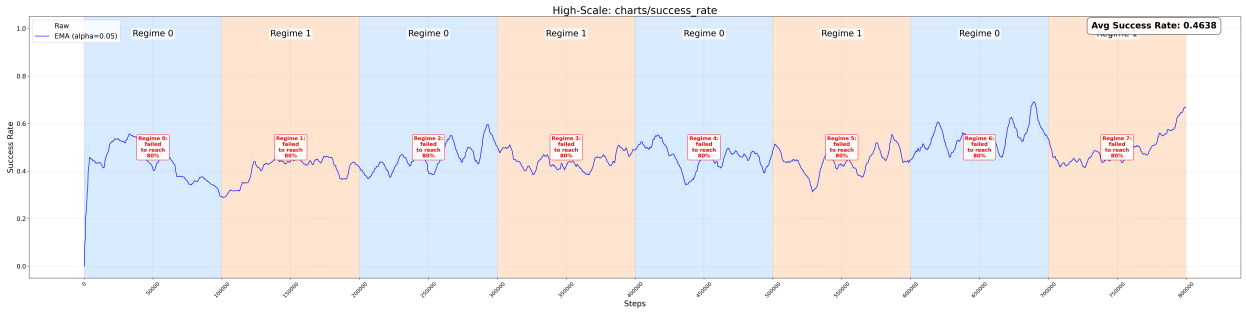


Fig. 8: Fast-switch static baseline.

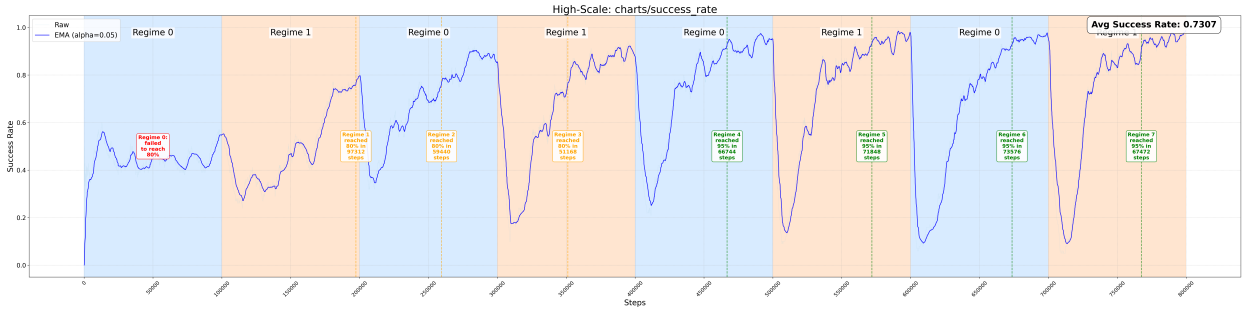


Fig. 9: Brain controller with scalar hyperparameter control.

A.2 Heuristic Neuromodulation vs Brain Controlled Neuromodulation

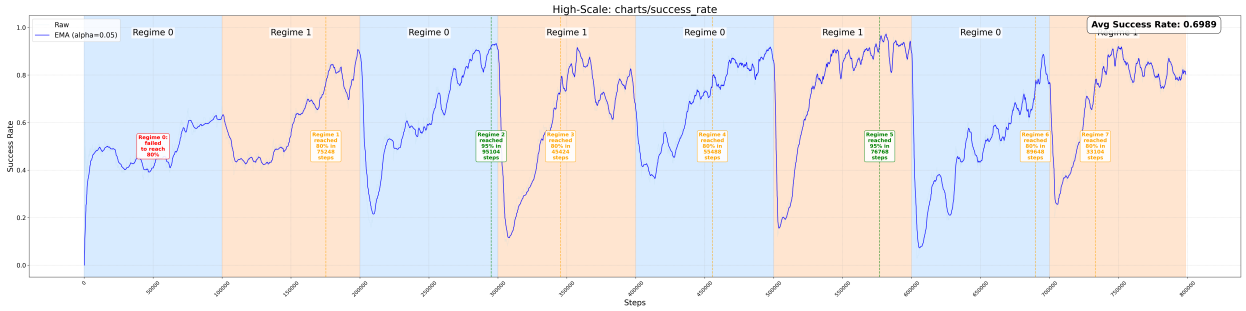


Fig. 10: Heuristic neuromodulation.

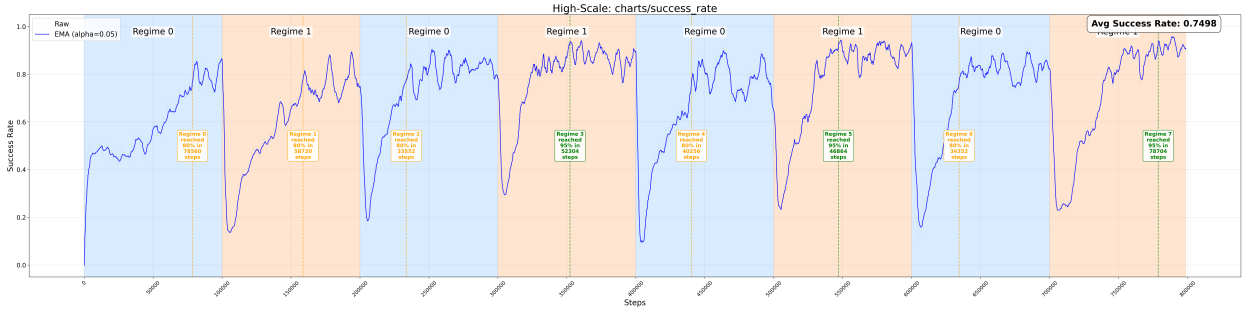


Fig. 11: Learned neuromodulation.

REFERENCES

[Chevalier-Boisvert et al.(2018)] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. MiniGrid: A Minimalistic Gridworld Environment for Reinforcement Learning. *arXiv preprint arXiv:1807.06757* (2018).

[Kirkpatrick et al.(2017)] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526.

[Pathak et al.(2017)] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017).

[Schulman et al.(2017)] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. In *arXiv preprint arXiv:1707.06347*.

[Sutton(1990)] Richard S. Sutton. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML)*.

TEAM CONTRIBUTIONS

Adam Gans
Designed high-level project plan, debating the best testbed and baselines to use. Designed and created much of the Regime-Specific Mixture of World Models approach. Designed and created most of the Context-Aware Transformer World Model approach. Researched the latest information and papers on continual learning and catastrophic forgetting. Also attempted to create the baseline world model with DreamerV3, but it ended up being far too powerful for the task to be helpful.

Elijah Tabachnik
Designed and implemented the Brain-based meta-learning component of the project. This work included building the `MetaEnv` outer-loop training setup, defining the Brain's observation and action spaces, integrating control over inner-loop learning levers, and implementing the context-code-based feature gating system inside the inner CNN actor-critic. Also developed the logging and analysis pipeline for neuromodulation diagnostics, generated and interpreted the corresponding plots, and wrote the sections of the paper describing the Brain architecture, neuromodulation mechanism, and its empirical effect on policy and value behavior across regime switches.

Juan Martinez
Designed and implemented the exploratory multi-head actor-critic architecture with EWC regularization that was included in the other approaches section of the report. Worked with Adam on the Regime-Specific Mixture of World Models approach. He also wrote most of the sections of the paper, excluding anything involving the regime-specific world model and meta-learning aspect of the project. This also included training a full PPO and Dyna-PPO model, plotting the results, and interpreting the logs and plot for the paper.

Harshvardhan Sharaff
I worked on the final project report, including writing and formatting the project template used for the submission. For the experiments, I implemented and ran the baseline training runs for the passive PPO and Dyna-PPO configurations on the MiniGrid-DualGoal environment, analyzed the resulting training curves, and produced the figures included in the report. I also prepared and developed the slides for the class presentation, covering the project motivation, approach, and experimental results.