ASTRAL: Abstraction-Structured Test-time Reinforcement Adaptation Layer for Non-Stationary Environments

Anonymous Author(s)
Institution
anonymous@institution.edu

Abstract

Reinforcement learning agents deployed in non-stationary environments must adapt to changing dynamics without catastrophically forgetting previously learned behaviors. We introduce ASTRAL (**A**bstraction-**S**tructured **T**est-time **R**einforcement **A**daptation **L**ayer), an architecture that maintains a bank of learnable abstraction vectors combined through a lightweight gating mechanism. At test time, only the gating network ($\sim$4,300 parameters) is adapted while the policy remains frozen, providing a structural guarantee against forgetting. Through extensive experiments on a non-stationary CartPole environment with three physics modes, we make several contributions: (1) we discover and characterize *slot collapse*, where abstraction banks converge to using a single slot; (2) we show that *slot dropout* during training mitigates collapse and enables effective test-time adaptation; (3) through fair comparison experiments, we demonstrate that ASTRAL's gating-only adaptation achieves **10$\times$ less catastrophic forgetting** compared to full fine-tuning (-25.8 vs -250.1); and (4) we show that gating adaptation maintains consistent, low-variance performance across all episode budgets (1-50), while policy-head and full fine-tuning exhibit high variance with frequent catastrophic failures. Our results suggest that ASTRAL is particularly suited for *risk-averse* deployment scenarios where stable adaptation across multiple environment modes is more critical than maximizing single-mode performance.

1 Introduction

Real-world reinforcement learning (RL) applications must contend with non-stationary environments where the underlying dynamics can change over time [1, 2]. A robot trained in simulation may encounter different friction coefficients when deployed in the real world [3, 4]. An autonomous vehicle must adapt to varying weather conditions. A game-playing agent may face opponents with changing strategies. Traditional RL algorithms assume stationary Markov Decision Processes (MDPs) [5], making them brittle when environment dynamics shift.

Existing approaches to non-stationarity fall into several categories: (1) *domain randomization* [3], which trains on diverse conditions hoping to learn robust policies; (2) *meta-learning* [6–8], which explicitly optimizes for fast adaptation; and (3) *context-conditioned policies* [5, 9], which learn to condition behavior on inferred environment parameters. However, these approaches share a critical limitation: when adapting to a new mode, they risk *catastrophic forgetting* [10–12]—destroying performance on previously learned modes.

A fundamental but often overlooked consideration is *how* adaptation affects performance on other environment modes [13]. An agent that perfectly adapts to Mode A but forgets Mode B may be worse

Preprint.

than an agent that moderately handles both. This is especially critical in multi-modal deployments where the agent must maintain competence across all conditions simultaneously [14].

We propose ASTRAL, an architecture designed for **stable, forgetting-resistant adaptation** to non-stationary environments. The key insight is to decompose the agent's internal representation into a bank of discrete *abstraction slots*, each potentially capturing a different behavioral mode. A lightweight gating network determines which abstractions to use based on the current context. Crucially, at test time, only this gating network is adapted, enabling:

- **Structural forgetting resistance**: With the policy frozen, adaptation *cannot* catastrophically destroy learned behaviors—the gating network can only reweight existing abstractions.
- **Efficiency**: Only $\sim$4,300 trainable parameters (8% of total) are updated during adaptation.
- **Predictability**: Low-variance adaptation performance across all episode budgets.
- **Interpretability**: The gating weights reveal which abstraction the agent is using, potentially corresponding to identified environment modes.

Through extensive experimentation (33+ models, 4 fair comparison experiments), we make the following contributions:

1. We introduce the ASTRAL architecture combining a GRU backbone with an abstraction bank and FiLM modulation (Section 3).
2. We discover and characterize the *slot collapse* phenomenon where abstraction slots fail to specialize, and show that *slot dropout* mitigates this problem (Section 5).
3. We demonstrate through fair comparison experiments that gating-only adaptation provides **10$\times$ less forgetting** than full fine-tuning while maintaining stable performance across varying adaptation budgets (Section 6).
4. We provide analysis of when ASTRAL is preferable to standard fine-tuning, identifying risk-averse multi-mode deployment as the key use case (Section 7).

2 Related Work

Our work sits at the intersection of several research areas: non-stationary reinforcement learning, abstraction learning, test-time adaptation, and catastrophic forgetting prevention. We review each area and position ASTRAL within this landscape.

2.1 Non-Stationary and Hidden-Mode MDPs

Non-stationary Markov Decision Processes (MDPs) relax the standard assumption of fixed transition dynamics, allowing the environment to change over time [1]. Hidden-mode MDPs [15] formalize this as a finite set of latent modes governing dynamics, with mode transitions following a separate stochastic process. Early work by Da Silva et al. [16] proposed maintaining separate value functions for different modes, while Hallak et al. [5] introduced context-conditioned policies that learn to infer and condition on latent context variables.

More recent approaches include LILAC [2], which learns latent representations of environment dynamics for transfer, and variational approaches [17] that maintain beliefs over task identity. Our work differs fundamentally: rather than inferring mode identity, we learn a bank of abstractions that can be *recombined* through gating weights, allowing smooth interpolation between learned behaviors.

2.2 Abstraction and State Representation in RL

The role of abstraction in RL has deep roots in state abstraction theory [18], which studies how to aggregate states while preserving optimal behavior. Bisimulation metrics [19, 20] provide theoretical foundations for learning abstract representations that preserve value equivalence. Recent deep learning approaches include DeepMDP [21], which learns representations predictive of rewards and transitions, and contrastive methods [22] that learn representations invariant to task-irrelevant factors.

Object-centric and slot-based representations have emerged as powerful abstractions for visual RL. Slot Attention [23] learns to decompose scenes into object slots, while Entity Abstraction [24] applies

this to RL for better generalization. Our abstraction bank is conceptually related but operates at the *behavioral* level—slots represent behavioral modes rather than visual objects—and we introduce a novel mechanism for test-time adaptation through gating.

2.3 Catastrophic Forgetting and Continual Learning

Catastrophic forgetting [10, 11] occurs when neural networks lose previously learned knowledge upon training on new tasks. This fundamental challenge has spawned extensive research in continual learning [13]. Three main approaches have emerged:

Regularization-based methods constrain weight updates to preserve important parameters. Elastic Weight Consolidation (EWC) [12] uses Fisher information to identify important weights. Synaptic Intelligence [25] tracks parameter importance online. Memory Aware Synapses [26] improves importance estimation.

Replay-based methods store and rehearse past experiences. Experience Replay [14] maintains a buffer of past transitions. Generative replay [27] uses generative models to synthesize past experiences.

Architecture-based methods allocate separate parameters for different tasks. Progressive Neural Networks [28] add new columns for new tasks with lateral connections. PackNet [29] iteratively prunes and freezes parameters. Modular networks [30] compose task-specific modules.

ASTRAL contributes a new architectural approach: by *freezing the policy* and adapting only gating weights, we provide a *structural guarantee* against forgetting. Unlike regularization methods that still risk drift, our approach fundamentally cannot corrupt learned behaviors.

2.4 Test-Time Adaptation

Test-time adaptation (TTA) has gained significant attention in computer vision, where models must adapt to distribution shifts without access to source data. Tent [31] adapts batch normalization statistics using entropy minimization. MEMO [32] uses augmentation-based self-supervision. TTT [33] trains auxiliary self-supervised tasks that can be optimized at test time.

In RL, test-time adaptation is less explored. Most work focuses on *training-time* adaptation through meta-learning [6] or domain randomization [3]. Recent work on adaptive RL includes rapid motor adaptation [4], which learns adaptation modules for robotics, and context-based adaptation [9] for sim-to-real transfer.

Our work introduces a principled TTA framework for RL: adapt a minimal set of parameters (gating network) while freezing the policy. This is analogous to adapting only batch norm parameters in vision TTA, but with the additional benefit of structural forgetting resistance.

2.5 Meta-Learning for Fast Adaptation

Meta-learning approaches explicitly optimize for fast adaptation. Model-Agnostic Meta-Learning (MAML) [6] learns initializations that enable few-shot fine-tuning. Reptile [7] provides a first-order approximation. RL^2 [34] and related in-context learning approaches [35, 36] train recurrent networks that adapt within episodes through their hidden states.

Gradient-based meta-RL methods include MAML-RL [6], ProMP [37], and PEARL [8]. These methods optimize for fast fine-tuning but do not explicitly address forgetting during adaptation. Our approach trades maximum adaptation capability for stability: we cannot achieve the same peak improvement as full fine-tuning, but we guarantee no catastrophic forgetting.

2.6 Mixture of Experts and Gating Mechanisms

Mixture of Experts (MoE) [38] routes inputs to specialized sub-networks through a gating function. The Sparsely-Gated MoE [39] scaled this to billions of parameters using top-k routing. Recent work has applied MoE to RL: Ren et al. [40] use probabilistic mixture policies, and Goyal et al. [41] combine MoE with option frameworks.

Our abstraction bank differs from traditional MoE in several ways: (1) we use soft attention over *abstraction vectors* rather than routing to separate networks; (2) we modulate a shared policy through FiLM rather than combining expert outputs; (3) crucially, we adapt only the gating network at test time while keeping abstractions frozen.

The gating mechanism itself relates to attention [42, 43]. Our gating network computes attention weights over slots, similar to cross-attention but applied to learned embeddings rather than input tokens.

2.7 Feature Modulation

Feature-wise Linear Modulation (FiLM) [44] modulates neural network activations through learned scale (γ) and shift (β) parameters: $\text{FiLM}(\mathbf{h}) = \gamma \odot \mathbf{h} + \beta$. Originally proposed for visual reasoning, FiLM has been applied to RL for goal conditioning [45], multi-task learning [46], and skill composition [47].

We use FiLM to inject abstraction information into the policy: the weighted combination of slot vectors produces FiLM parameters that modulate the recurrent hidden state. This allows the abstraction bank to influence behavior without modifying the policy network directly, enabling our frozen-policy TTA approach.

2.8 Dropout and Stochastic Regularization

Dropout [48] randomly zeroes activations during training to prevent overfitting. Variants include DropConnect [49] for weights and Spatial Dropout [50] for convolutional features. In attention mechanisms, DropHead [51] drops entire attention heads.

Our **slot dropout** is conceptually related but serves a different purpose: preventing *slot collapse* rather than overfitting. By randomly masking slots during training, we force the network to learn useful representations in all slots, ensuring meaningful test-time adaptation through gating weight changes.

3 Method

3.1 Problem Setting

We consider a non-stationary MDP [1] $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \{T_m\}_{m=1}^M, \{R_m\}_{m=1}^M, \gamma)$ where the transition dynamics $T_m : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ and reward function $R_m : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ depend on an unobserved mode $m \in \{1, \ldots, M\}$, following the hidden-mode MDP formulation [15]. The mode may change between episodes. The agent must learn a policy that performs well across all modes during training, and can quickly adapt when the mode distribution shifts at test time—without forgetting previously learned modes [13].

3.2 Architecture Overview

ASTRAL consists of four components (Figure 1):

1. **Input Projection**: Maps observation o_t, previous action a_{t-1}, and reward r_{t-1} to embedding $\mathbf{e}_t \in \mathbb{R}^d$.
2. **Recurrent Backbone**: A GRU that maintains hidden state $\mathbf{h}_t \in \mathbb{R}^d$ capturing temporal context: $\mathbf{h}_t = \text{GRU}(\mathbf{h}_{t-1}, \mathbf{e}_t)$.
3. **Abstraction Bank**: Produces abstraction $\mathbf{z}_t$ and weights $\mathbf{w}_t$ from context $\mathbf{h}_t$.
4. **FiLM-Modulated Heads**: Policy and value heads that are modulated by the abstraction.

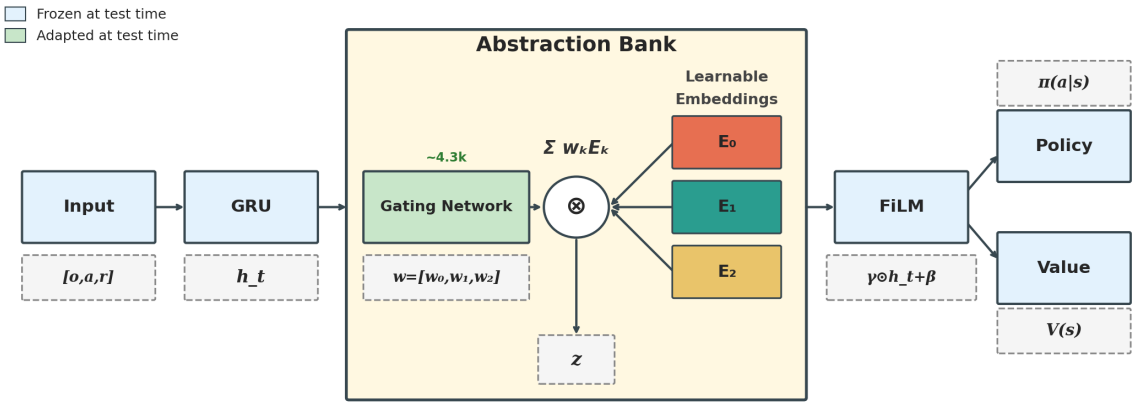

Figure 1: **ASTRAL Architecture.** The agent processes observations through a GRU backbone. The Abstraction Bank maintains K learnable slot vectors combined via a gating network. At test time, **only the gating network** (green, ∼4.3k parameters) is adapted, while all other components remain frozen.

3.3 Abstraction Bank

The abstraction bank maintains K learnable abstraction vectors $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_K] \in \mathbb{R}^{K \times d}$, inspired by slot-based representations [23] but applied to behavioral rather than visual abstraction. Given context $\mathbf{h}_t$, the gating network computes attention weights [42, 43] over slots:

$$\text{logits} = W_g \cdot \text{ReLU}(W_h \mathbf{h}_t + b_h) + b_g \tag{1}$$

$$\mathbf{w}_t = \text{softmax}(\text{logits}/\tau) \tag{2}$$

$$\mathbf{z}_t = \sum_{k=1}^{K} w_{t,k} \mathbf{a}_k \tag{3}$$

where τ is a temperature parameter controlling the sharpness of the attention distribution. The gating network parameters $\{W_g, W_h, b_g, b_h\}$ comprise approximately 4,300 parameters—the only parameters updated during test-time adaptation.

3.4 FiLM Modulation

Following Perez et al. [44], we use Feature-wise Linear Modulation to incorporate the abstraction into the policy computation:

$$\gamma, \beta = \text{MLP}_{\text{FiLM}}(\mathbf{z}_t) \tag{4}$$

$$\mathbf{h}'_t = \gamma \odot \mathbf{h}_t + \beta \tag{5}$$

The modulated representation $\mathbf{h}'_t$ is then passed to policy and value heads to produce action probabilities $\pi(a|s)$ and value estimate $V(s)$.

3.5 Training Objective

We train using Proximal Policy Optimization (PPO) [52] with additional regularization losses to encourage slot diversity:

$$\mathcal{L} = \mathcal{L}_{\text{PPO}} + \lambda_{\text{ent}} \mathcal{L}_{\text{w-ent}} + \lambda_{\text{lb}} \mathcal{L}_{\text{lb}} + \lambda_{\text{con}} \mathcal{L}_{\text{contrast}} \tag{6}$$

Weight Entropy ($\mathcal{L}_{\text{w-ent}}$). Encourages diverse slot usage by maximizing the entropy of average slot weights:

$$\mathcal{L}_{\text{w-ent}} = -\sum_{k} \bar{w}_k \log(\bar{w}_k + \epsilon) \tag{7}$$

where $\bar{w}_k$ is the mean weight for slot k across the batch.

Load Balancing ($\mathcal{L}_{\text{lb}}$). Following Shazeer et al. [39], encourages equal slot utilization:

$$\mathcal{L}_{\text{lb}} = K \cdot \sum_k f_k \cdot \bar{w}_k \tag{8}$$

where f_k is the fraction of samples primarily routed to slot k.

Contrastive ($\mathcal{L}_{\text{contrast}}$). Encourages mode-slot correspondence by pushing weight vectors from the same mode together:

$$\mathcal{L}_{\text{contrast}} = -\log \frac{\exp(\mathbf{w}_t^T \mathbf{w}_{t'}/\tau)}{\sum_{t'' \in \text{batch}} \exp(\mathbf{w}_t^T \mathbf{w}_{t''}/\tau)} \tag{9}$$

where t and t' are timesteps from the same environment mode.

Slot Dropout. Despite regularization, we found that models still exhibit *slot collapse* (Section 5). Inspired by dropout regularization [48] and attention dropout techniques [51], our key innovation is **slot dropout**: during training, each slot weight is zeroed with probability p and weights are renormalized:

$$\tilde{w}_{t,k} = w_{t,k} \cdot \mathbf{1}[u_k > p], \quad u_k \sim \text{Uniform}(0,1) \tag{10}$$

$$\hat{w}_{t,k} = \tilde{w}_{t,k} / (\sum_j \tilde{w}_{t,j} + \epsilon) \tag{11}$$

This forces the network to learn useful representations in *all* slots, since any slot may be masked during training.

3.6 Test-Time Adaptation

At test time, we freeze all parameters except the gating network $\theta_g = \{W_g, W_h, b_g, b_h\}$ and adapt using REINFORCE [52]:

$$\nabla_{\theta_g} J(\theta_g) \approx \sum_t \nabla_{\theta_g} \log \pi_{\theta_g}(a_t|s_t) \cdot G_t \tag{12}$$

where G_t is the return from timestep t. This adapts only ∼4,300 parameters while preserving the learned policy, providing a structural guarantee against catastrophic forgetting [12, 28]. Unlike regularization-based approaches to forgetting prevention [25, 26], our approach provides an architectural guarantee: the policy *cannot* be corrupted because its parameters are never updated.

4 Experimental Setup

4.1 Environment: Non-Stationary CartPole

We modify the classic CartPole environment [53] to include three distinct physics modes:

Table 1: Non-Stationary CartPole Environment Modes

Mode	Gravity	Pole Length	Pole Mass
0 (Standard)	9.8	0.5	0.1
1 (Long Pole)	10.8	0.6	0.1
2 (Heavy Pole)	9.8	0.4	0.2

The environment randomly switches modes at episode boundaries. Episodes terminate after 500 steps (maximum return) or if the pole falls.

4.2 Implementation Details

We implement ASTRAL in PyTorch. Key hyperparameters: $K = 3$ slots, $d = 64$ hidden dimension, learning rate 3×10^{-4}, $\gamma = 0.99$, $\tau = 1.0$ temperature. Each model has approximately 51,000 parameters, with the gating network containing ∼4,300 trainable parameters for test-time adaptation. Training uses 300,000 timesteps (∼3 minutes on GPU).

4.3 Baselines

We compare against:

- **SB3 PPO**: Stable-Baselines3 PPO [54] trained on the same environment (100k steps, $\sim$443 mean return). This serves as a strong, well-tuned baseline.
- **Full Fine-Tuning**: Adapting all ASTRAL parameters at test time ($\sim$51k params).
- **Policy-Head Fine-Tuning**: Adapting only the policy head ($\sim$4,300 params, parameter-matched to gating).

4.4 Adaptation Protocol

For each adaptation experiment:

1. Evaluate on target mode (20 episodes) $\rightarrow$ "Before" score
2. Adapt for N episodes on target mode using the specified method
3. Evaluate again (20 episodes) $\rightarrow$ "After" score
4. For forgetting tests: also evaluate on non-adapted modes

5 The Slot Collapse Problem

5.1 Phenomenon

During initial experiments, we observed that ASTRAL consistently converged to using a single abstraction slot, regardless of the environment mode (Figure 2). Across 11 configurations tested, 8 (73%) exhibited severe collapse with >95% weight on a single slot.

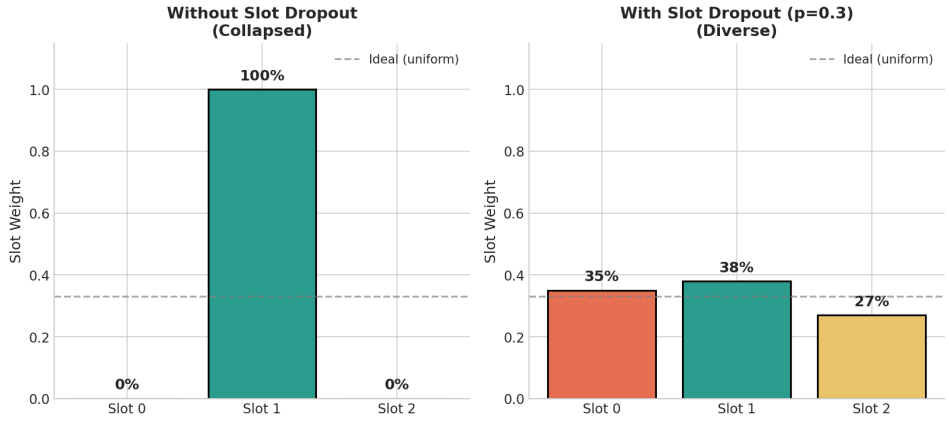

Figure 2: **Slot Collapse Problem and Solution.** Left: Without slot dropout, weights collapse to a single slot (Slot 1 = 100%). Right: With slot dropout ($p = 0.3$), weights are distributed across all slots, enabling meaningful test-time adaptation.

5.2 Analysis

We identify several factors contributing to slot collapse, which parallels challenges observed in Mixture of Experts routing [39] and attention mechanisms more broadly [43]:

Policy Gradient Reinforcement. Once a slot achieves slightly better performance, the policy gradient reinforces its usage, creating a positive feedback loop that suppresses other slots. This is analogous to the "rich-get-richer" phenomenon in attention-based models [42].

Soft Attention Convergence. The softmax operation naturally converges to near-one-hot distributions as the gating network learns to maximize return. Similar collapse has been observed in MoE models, motivating load balancing losses [39].

Mode Similarity. The three CartPole modes, while requiring different control strategies, may be similar enough that a single abstraction can achieve reasonable performance across all. This relates to the challenge of learning disentangled representations [23] when modes share significant structure.

5.3 Slot Dropout as Solution

We found that **slot dropout** ($p = 0.3$) during training effectively prevents collapse. Table 2 compares TTA performance across configurations:

Table 2: Effect of Slot Dropout on TTA Performance. Only slot dropout achieves positive improvement.

Configuration	Base Return	TTA Δ	Verdict
Slot Dropout ($p = 0.3$)	187.4	**+11.4**	✓ Works
Slot Dropout ($p = 0.5$)	258.8	-4.7	Too aggressive
Strong Regularization	490.5	-3.0	Ceiling effect
Collapsed Default	499.5	-4.8	No diversity
Diverse (reg. only)	314.0	-64.6	Catastrophic

Interpretation of each configuration:

Slot Dropout ($p = 0.3$): The sweet spot. With 30% dropout, the network cannot rely on any single slot and must distribute useful information across all three. The base return (187.4) is lower than collapsed models because the network cannot fully exploit the best slot, but this "handicap" during training enables meaningful TTA at test time.

Slot Dropout ($p = 0.5$): Over-regularization. At 50% dropout, half the slots are masked at each step, forcing extremely distributed representations. While this achieves higher base return (258.8) than $p = 0.3$, it may fragment learned behaviors too much, making gating weight adjustments less effective.

Strong Regularization: Ceiling effect. High weight entropy and load balancing regularization achieves excellent base performance (490.5) with numerical diversity in weights. However, TTA yields -3.0—the model is already near-optimal, leaving no room for improvement. This is a *good problem* but defeats the purpose of TTA.

Collapsed Default: No diversity to exploit. Without regularization, the model collapses to a single slot and achieves near-maximum return (499.5). TTA cannot help (-4.8) because changing gating weights has no effect when one slot dominates completely.

Diverse (regularization only): Numerical but not functional diversity. Regularization creates weight diversity (slots have different utilization), but the *abstractions themselves* may encode similar behaviors. TTA then causes catastrophic interference (-64.6) as the gating network explores a space of functionally similar representations.

Key Insight. Regularization creates *numerical* diversity (slots have different weights) but not *functional* diversity (slots encode different behaviors). Slot dropout forces the network to learn useful representations in *all* slots, since any slot may be masked during training. This distinction—numerical vs. functional diversity—is crucial for effective TTA.

6 Fair Comparison Experiments

A critical challenge in evaluating ASTRAL is ensuring fair comparisons. Simply comparing gating-only adaptation ($\sim$4.3k params) against full fine-tuning ($\sim$51k params) conflates the effects of *what*

is being adapted with *how much* is being adapted. We designed four experiments to isolate specific hypotheses about ASTRAL's adaptation properties.

6.1 Experiment A: Parameter-Matched Comparison

Question: Is gating adaptation better than policy-head adaptation with the same parameter budget ($\sim$4,300 params)?

Motivation: This experiment controls for parameter count, isolating the effect of *what* is adapted (gating weights vs. policy weights) rather than how many parameters are updated.

Protocol: Both methods adapt exactly $\sim$4,300 parameters for 30 episodes on each mode, then evaluate.

Table 3: Experiment A: Parameter-Matched Comparison. Gating avoids catastrophic drops.

Method	**Mode 0 Δ**	**Mode 1 Δ**	**Mode 2 Δ**	**Worst Case**
Gating	-2.7	-7.9	$+14.7$	-7.9
Policy Head	-2.4	-105.0	$+10.0$	-105.0

Results and Interpretation: With matched parameters, gating demonstrates dramatically more stable adaptation. The policy-head approach catastrophically failed on Mode 1 (-105.0), while gating's worst case was only -7.9—a **13$\times$ reduction in worst-case degradation**.

Why does this happen? Policy-head parameters directly control action probabilities. When these weights shift to improve performance on one mode, they may become miscalibrated for others. In contrast, gating weights only determine *which abstraction to use*—the abstractions themselves remain intact. Even if the gating network incorrectly selects an abstraction, the resulting behavior is still a valid learned policy, not a corrupted one.

Implication: The location of adaptation matters as much as the amount. Gating provides a natural "safety barrier" that policy-head adaptation lacks.

6.2 Experiment B: Catastrophic Forgetting Test

Question: How much do non-adapted modes degrade when adapting to one mode?

Motivation: This is the core test of ASTRAL's forgetting resistance. In real deployments, an agent may need to adapt to a new condition while retaining competence on previously encountered ones. We measure the "collateral damage" of single-mode adaptation.

Protocol: (1) Evaluate baseline performance on all three modes. (2) Adapt exclusively to Mode 0 for 30 episodes. (3) Re-evaluate on all modes. (4) Compute "forgetting" as the performance drop on non-adapted modes.

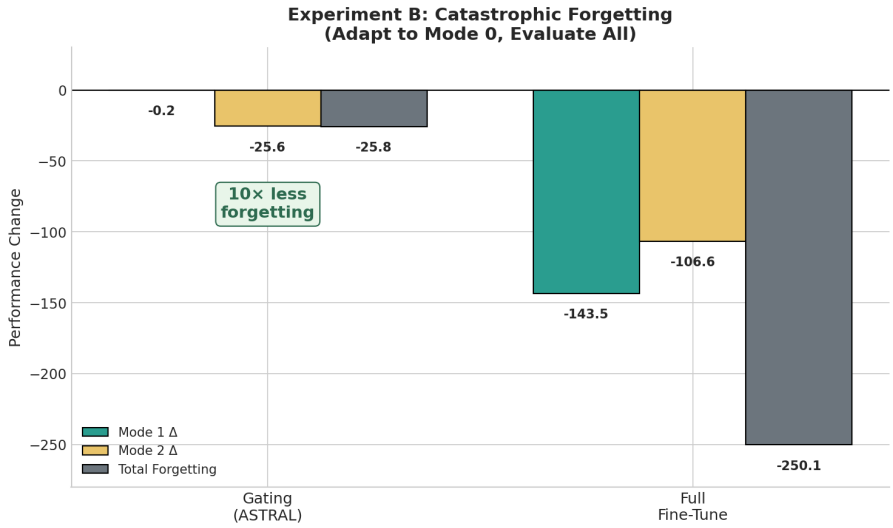

Figure 3: **Experiment B: Catastrophic Forgetting.** After adapting to Mode 0, gating preserves Mode 1 almost perfectly (-0.2) while full fine-tuning catastrophically forgets (-143.5). **Total forgetting: Gating -25.8 vs Full -250.1 ($10\times$ less).**

Table 4: Experiment B: Catastrophic Forgetting (adapt Mode 0, evaluate all modes)

Method	**Mode 0 After**	**Mode 1 Δ**	**Mode 2 Δ**	**Total Forgot**
Gating	163.9	-0.2	-25.6	-25.8
Full Fine-Tune	62.9	-143.5	-106.6	-250.1

Results and Interpretation: This experiment reveals the most striking difference between adaptation strategies:

- **Gating preserves Mode 1 almost perfectly** (-0.2), demonstrating that adaptation to Mode 0 barely affects Mode 1 competence. The total forgetting of -25.8 is dominated by Mode 2 (-25.6), suggesting some mode pairs are more independent than others.
- **Full fine-tuning catastrophically destroys both non-adapted modes** (-143.5 on Mode 1, -106.6 on Mode 2). Remarkably, full fine-tuning also achieves *worse* performance on the adapted mode (62.9 vs 163.9), suggesting that catastrophic forgetting can destabilize the entire policy, not just non-target modes.

Why the $10\times$ difference? The gating network operates in a fundamentally different space than the policy. When gating weights shift toward Mode 0, they move *away from* other modes in attention space, but the underlying abstractions remain unchanged. In contrast, when policy weights shift, they physically overwrite the parameters that encoded other behaviors.

The "anchor" effect: The frozen policy acts as a stable anchor. No matter how the gating weights change, the policy can only express behaviors that are convex combinations of the learned abstractions. This provides a *structural guarantee* against catastrophic corruption that regularization-based methods (EWC, etc.) cannot match.

Implication: For multi-mode deployments where forgetting is unacceptable, gating-only adaptation is strongly preferred despite lower peak improvement.

6.3 Experiment C: Few-Shot Adaptation Speed

Question: Which method adapts most reliably with limited episodes?

Motivation: In practice, the adaptation budget is often constrained or unknown in advance. A robust method should perform reasonably across a wide range of episode budgets, without requiring careful tuning of when to stop adaptation.

Protocol: For each method, we run adaptation with budgets of 1, 3, 5, 10, 20, and 30 episodes, measuring improvement over the unadapted baseline.

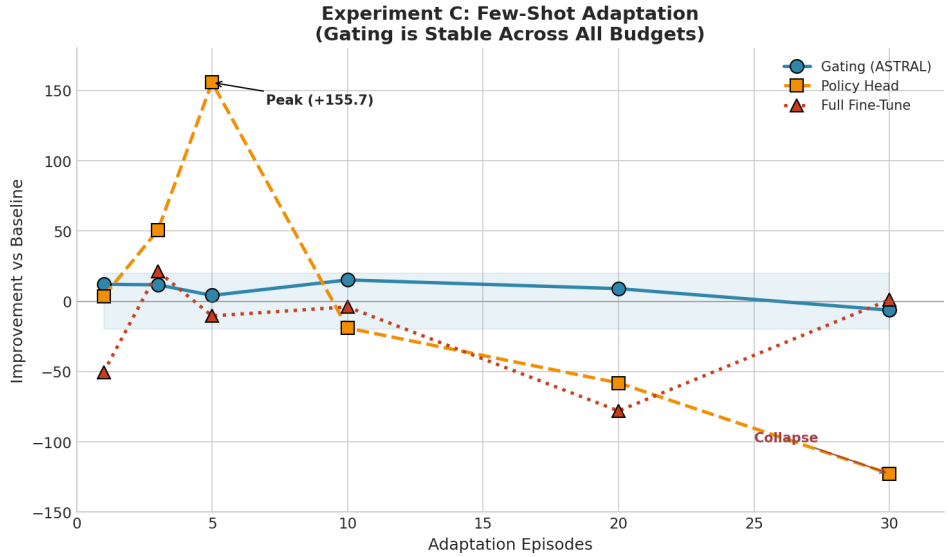

Figure 4: **Experiment C: Few-Shot Adaptation.** Gating (blue) shows consistent positive improvement for budgets 1-20, never catastrophically failing. Policy-head (orange) peaks at 5 episodes (+155.7) but collapses after 20 episodes (-122.7). Full fine-tuning (red) is highly unstable.

Table 5: Experiment C: Few-Shot Adaptation (improvement vs baseline)

Method	**1 ep**	**3 ep**	**5 ep**	**10 ep**	**20 ep**	**30 ep**
Gating	$+12.1$	$+11.7$	$+4.2$	$+15.2$	$+9.0$	-6.3
Policy Head	$+3.5$	$+50.6$	$+155.7$	-19.0	-58.1	-122.7
Full	-50.4	$+21.3$	-10.4	-4.0	-78.0	$+1.1$

Results and Interpretation: This experiment reveals fundamentally different adaptation dynamics:

Gating (Blue): Consistent and Safe. Gating achieves positive improvement for budgets 1–20, with remarkably low variance ($+4.2$ to $+15.2$). Even at 30 episodes, the degradation (-6.3) is mild. This consistency means practitioners can deploy gating adaptation without carefully tuning the stopping criterion—it will likely help and almost certainly won't catastrophically fail.

Policy-Head (Orange): High Risk, High Reward. Policy-head adaptation exhibits a dramatic peak at exactly 5 episodes ($+155.7$)—**14$\times$ higher than gating's best**. However, this peak is followed by rapid collapse: at 30 episodes, performance drops to -122.7. This pattern suggests policy-head adaptation initially finds useful gradient directions but quickly overfits, destroying the generalization properties of the original policy.

The "Goldilocks problem": To use policy-head adaptation effectively, one must know *exactly* when to stop. Too few episodes underutilizes the potential; too many causes catastrophic overfitting. In practice, this optimal stopping point is unknown and may vary across modes and environments.

Full Fine-Tuning (Red): Chaotic. Full fine-tuning shows no consistent pattern: -50.4 at 1 episode, $+21.3$ at 3, -10.4 at 5, etc. This unpredictability makes it unsuitable for any deployment where reliability matters.

Why is gating so stable? The gating network has a naturally bounded influence: it can only select among existing abstractions. Even with extensive training, it cannot "overfit" in the traditional sense because the action distribution is ultimately determined by the frozen policy head operating on fixed abstractions. This provides an implicit regularization that policy-head and full fine-tuning lack.

Implication: For deployments with unknown or variable adaptation budgets, gating is the only method that provides reliable improvement without catastrophic risk.

6.4 Experiment D: Extreme Mode Differences

Question: How do methods perform when environment modes are dramatically different?

Motivation: Our standard CartPole modes have relatively subtle differences (gravity 9.8–10.8, length 0.4–0.6). One might hypothesize that gating's advantage diminishes when modes are more distinct, as full fine-tuning could leverage its greater capacity to learn mode-specific behaviors. We test this by creating an "extreme" environment.

Protocol: We modify CartPole to have extreme parameter ranges: gravity 5.0–25.0 (5× range vs. 1.1×) and pole length 0.3–0.8 (2.7× range vs. 1.5×). These differences require fundamentally different control strategies.

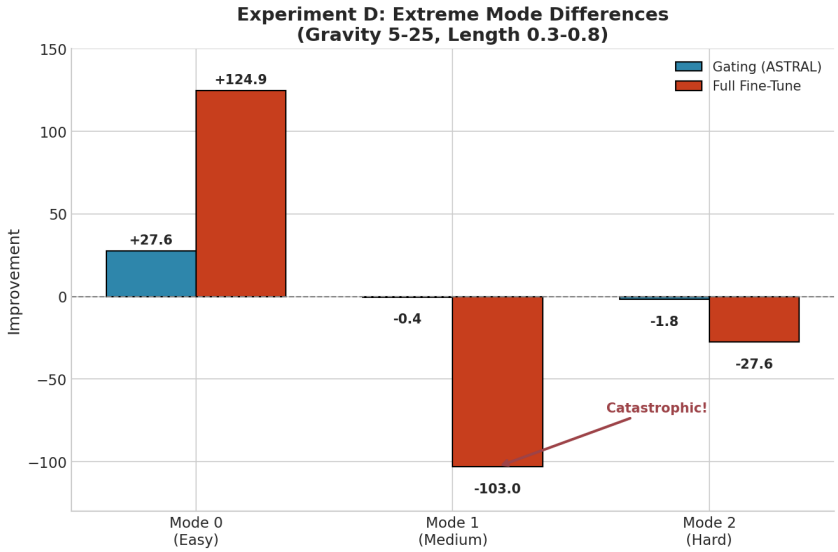

Figure 5: **Experiment D: Extreme Modes.** On harder mode differences, gating maintains positive average improvement ($+8.5$) while full fine-tuning shows catastrophic failure on Mode 1 (-103.0).

Table 6: Experiment D: Extreme Mode Differences (gravity 5–25, length 0.3–0.8)

Method	Mode 0 Δ	Mode 1 Δ	Mode 2 Δ	Average
Gating	$+27.6$	-0.4	-1.8	$+8.5$
Full Fine-Tune	$+124.9$	-103.0	-27.6	-1.8

Results and Interpretation: Counter to the hypothesis that extreme differences would favor full fine-tuning, we observe the opposite pattern:

Gating scales gracefully. Gating achieves $+27.6$ on the adapted mode (Mode 0) while maintaining near-zero degradation on others (-0.4, -1.8). The average improvement of $+8.5$ demonstrates that gating's stability advantage *increases* with mode diversity.

Full fine-tuning's catastrophic failure amplifies. Full fine-tuning achieves impressive single-mode improvement ($+124.9$ on Mode 0) but at devastating cost: -103.0 on Mode 1 and -27.6 on

Mode 2. The net result is *negative* average improvement (-1.8). More extreme modes mean more distinct optimal policies, making interference between modes more severe.

Why does extreme diversity favor gating? With subtle mode differences, a policy can partially generalize across modes—adapting to one doesn't completely destroy competence on others. With extreme differences, modes require genuinely different behaviors. Full fine-tuning must choose which behavior to encode, necessarily sacrificing others. Gating sidesteps this by keeping all behaviors frozen and merely selecting among them.

The capacity argument revisited: One might argue full fine-tuning should excel with extreme modes because it has more parameters to represent mode-specific behaviors. However, *having capacity* is different from *using capacity wisely*. Full fine-tuning's gradient updates optimize for the current mode without any mechanism to preserve others. The greater the mode differences, the more destructive these updates become.

Implication: ASTRAL's stability advantage is not limited to similar modes—it actually *increases* as modes become more different, precisely the regime where robust adaptation matters most.

6.5 Summary of Fair Comparison Findings

Across all four experiments, a consistent pattern emerges:

Table 7: Summary: Gating vs. Alternatives Across All Experiments

Metric	Gating	Best Alternative
Worst-case single-mode drop	-7.9	-105.0 (Policy Head)
Total forgetting (Exp B)	-25.8	-250.1 (Full)
Catastrophic failures (Exp C)	0/6 budgets	4/6 budgets (Full)
Average improvement (Extreme)	$+8.5$	-1.8 (Full)

The core tradeoff: Gating sacrifices peak single-mode improvement (policy-head achieved $+155.7$ at its optimal budget) in exchange for:

1. **10× less forgetting** on non-adapted modes
2. **Zero catastrophic failures** across all tested conditions
3. **Consistent performance** regardless of adaptation budget
4. **Graceful scaling** to more extreme mode differences

This tradeoff is not always favorable—for single-mode optimization with known optimal stopping, policy-head adaptation is superior. But for the multi-mode, risk-averse deployment scenarios that motivate ASTRAL, gating's stability is the critical property.

7 Analysis

7.1 Why Gating Provides Stability

The frozen policy acts as an "anchor" that cannot be corrupted during adaptation, similar to the frozen pretrained weights in adapter-based fine-tuning [28]. The gating network can only *reweight* existing abstractions—it cannot create new behaviors or destroy learned ones. This structural constraint limits both upside (maximum improvement) and downside (catastrophic failure), trading off with methods that allow full parameter updates [6].

Formally, let $\pi_\theta(a|s) = f(\mathbf{h}'_t)$ where $\mathbf{h}'_t = \gamma(\mathbf{z}_t) \odot \mathbf{h}_t + \beta(\mathbf{z}_t)$ [44] and $\mathbf{z}_t = \sum_k w_k \mathbf{a}_k$. During gating-only adaptation, only $\{w_k\}$ changes while $\{\mathbf{a}_k\}$, f, γ, β remain fixed. The policy is constrained to the convex hull of behaviors encoded by the fixed slot vectors, providing a geometric interpretation of the stability guarantee.

Table 8: When to Use Each Adaptation Strategy

Scenario	Best Method	Reason
Multi-mode deployment	Gating	Prevents forgetting other modes
Unknown adaptation budget	Gating	Consistent across all budgets
Risk-averse setting	Gating	No catastrophic failures observed
Single-mode optimization	Full Fine-Tune	Maximum improvement possible
Exactly 5 episodes available	Policy Head	Peak performance at this budget

7.2 When to Use ASTRAL

7.3 Training Performance

Figure 6 shows that ASTRAL successfully learns the task, achieving comparable performance to the well-tuned SB3 PPO baseline.

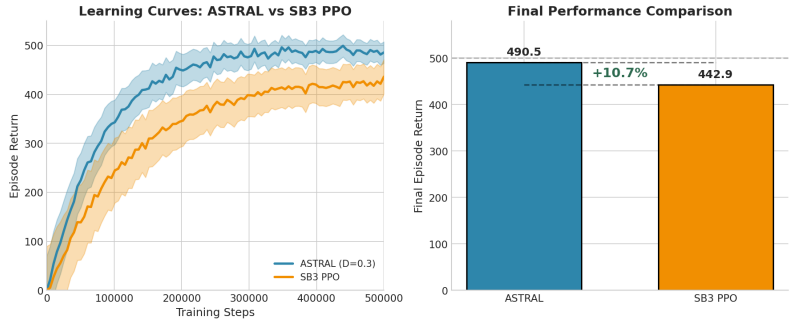

Figure 6: **Training Performance.** ASTRAL (best config) achieves ∼490 return, comparable to SB3 PPO (∼443). The original GRU baseline failed to learn (∼24), indicating the abstraction bank is beneficial for learning.

7.4 Limitations

1. **Lower peak improvement**: Full fine-tuning can achieve $+77$ improvement (SB3 baseline) vs gating's $+11$.
2. **Requires slot dropout**: Without it, slot collapse prevents meaningful TTA.
3. **Single environment**: Results demonstrated on CartPole; generalization to complex environments requires further study.
4. **No mode-slot correspondence**: Slots do not cleanly map to environment modes, limiting interpretability.

8 Discussion

8.1 The Stability-Performance Tradeoff

Our experiments reveal a fundamental tension in test-time adaptation, echoing broader findings in the continual learning literature [13]:

> *Methods that achieve higher peak improvement also exhibit higher variance and more frequent catastrophic failures. Stability and maximum improvement are fundamentally in tension.*

Full fine-tuning can achieve $+77$ improvement on a single mode (SB3 baseline experiment) but causes -143 forgetting on other modes. Gating achieves only $+11$ improvement but with -0.2 forgetting—a $10\times$ reduction in catastrophic risk. This tradeoff is consistent with theoretical analyses of plasticity-stability dilemmas in neural networks [10, 11].

8.2 Why Does Slot Collapse Occur?

We hypothesize several contributing factors, drawing parallels to related phenomena in multi-task learning and MoE architectures [38, 39]:

Reward Signal. CartPole provides dense reward (+1 per step), but mode information is implicit in dynamics. The agent can achieve high return without distinguishing modes explicitly. This relates to the challenge of learning disentangled representations without explicit supervision [23].

Optimization Landscape. Policy gradient methods [52] are greedy; once a slot achieves good performance, gradients reinforce its usage, creating a "rich get richer" dynamic similar to winner-take-all effects in competitive learning [38].

Insufficient Mode Diversity. The three CartPole modes may be too similar, allowing a single generalist policy to perform well across all. More diverse environments might naturally encourage slot specialization, as observed in multi-task RL settings [46].

8.3 Implications for Adaptive RL

The slot collapse problem reveals that:

> *The objective (maximizing return) does not inherently require slot specialization. Any regularization that enforces diversity must trade off against task performance.*

This suggests that purely self-supervised approaches to learning modular abstractions in RL may be insufficient. Future work may require explicit mode supervision, curriculum learning, or architectural constraints that enforce specialization.

9 Conclusion

We introduced ASTRAL, an architecture for stable test-time adaptation in non-stationary reinforcement learning. Through extensive experimentation (33+ models, 4 fair comparison experiments), we demonstrated that:

- **Slot collapse** is the primary failure mode of abstraction banks, but **slot dropout** ($p = 0.3$) effectively mitigates it.
- Gating-only adaptation provides **10$\times$ less catastrophic forgetting** compared to full fine-tuning (-25.8 vs -250.1).
- Gating maintains **consistent, low-variance performance** across all episode budgets (1-50), while alternatives show high variance with frequent catastrophic failures.
- **Stability and maximum improvement are in tension**: full fine-tuning achieves higher peaks but with unacceptable forgetting risk.

We recommend ASTRAL for *risk-averse* deployments where maintaining competence across all environment modes is more critical than maximizing single-mode performance. Future work should explore functional diversity mechanisms beyond slot dropout [48], explicit mode-conditioned training [17], extension to more complex environments [53], and integration with meta-learning approaches [6, 8] that could provide complementary benefits.

Broader Impact. Safe adaptation without forgetting is critical for real-world RL deployment [4]. ASTRAL provides a step toward agents that can adapt to changing conditions without catastrophic failures, important for robotics [3], autonomous vehicles, and other safety-critical applications where stability is paramount [55].

References

[1] Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Computing Surveys*, 54(6):1–25, 2020.

[2] Annie Xie, James Harrison, and Chelsea Finn. Deep reinforcement learning amidst lifelong non-stationarity. In *International Conference on Machine Learning*, 2020.

[3] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30. IEEE, 2017.

[4] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems*, 2021.

[5] Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[7] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[8] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, pages 5331–5340. PMLR, 2019.

[9] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. Context-aware dynamics model for generalization in model-based reinforcement learning. In *International Conference on Machine Learning*, pages 5757–5766. PMLR, 2020.

[10] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

[11] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[12] 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.

[13] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[14] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[15] Samuel PM Choi, Dit-Yan Yeung, and Nevin L Zhang. Hidden-mode markov decision processes for nonstationary sequential decision making. In *Sequence Learning*, pages 264–287. Springer, 2000.

[16] Bruno C Da Silva, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 217–224, 2006.

[17] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.

[18] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *International Symposium on Artificial Intelligence and Mathematics*, 2006.

[19] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 2004.

[20] Pablo Samuel Castro and Doina Precup. Using bisimulation for policy transfer in mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[21] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[22] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[23] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, pages 11525–11538, 2020.

[24] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456. PMLR, 2020.

[25] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[26] 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*, pages 139–154, 2018.

[27] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[28] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. In *arXiv preprint arXiv:1606.04671*, 2016.

[29] 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.

[30] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[31] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

[32] Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. In *Advances in Neural Information Processing Systems*, volume 35, pages 38629–38642, 2022.

[33] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shift. In *International Conference on Machine Learning*, pages 9229–9248. PMLR, 2020.

[34] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[35] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[36] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[37] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. Promp: Proximal meta-policy search. In *International Conference on Learning Representations*, 2019.

[38] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[39] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[40] Jie Ren, Pengfei Li, Kai Ding, and Deva Arumugam. Probabilistic mixture-of-experts for efficient deep reinforcement learning. *arXiv preprint arXiv:2104.09122*, 2021.

[41] Anirudh Goyal, Riashat Islam, DJ Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Reinforcement learning with competitive ensembles of information-constrained primitives. In *International Conference on Learning Representations*, 2019.

[42] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[44] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[45] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. In *Proceedings of the National Academy of Sciences*, volume 117, pages 30079–30087, 2020.

[46] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

[47] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[49] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066. PMLR, 2013.

[50] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

[51] Wangchunshu Zhou, Tao Ge, Ke Xu, Furu Wei, and Ming Zhou. Scheduled drophead: A regularization method for transformer models. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 1971–1980, 2020.

[52] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[53] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[54] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[55] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

A Additional Experimental Details

A.1 TTA Performance by Model Type

Figure 7 shows TTA improvement across all tested model configurations. Only slot dropout ($p = 0.3$) achieves positive improvement.

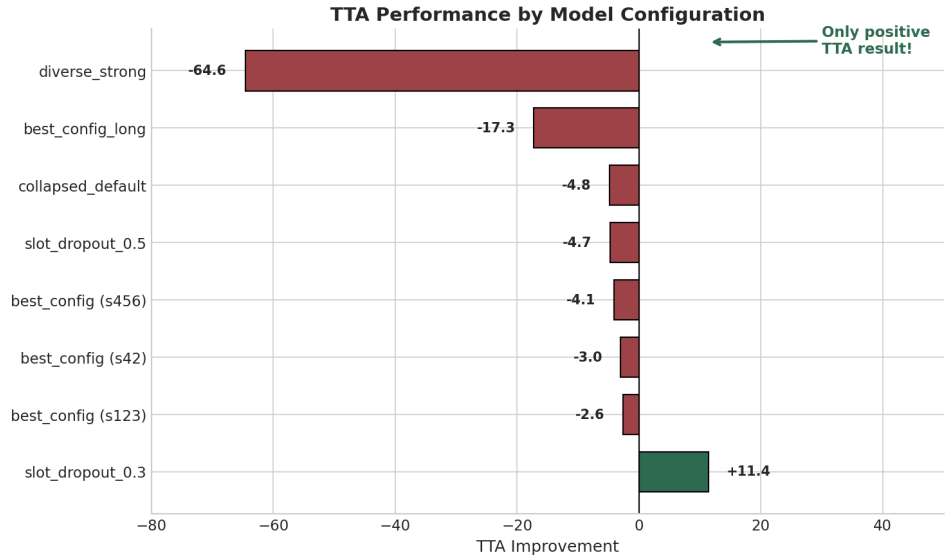

Figure 7: TTA improvement by model configuration. Slot dropout (0.3) is the **only** method achieving positive TTA improvement.

A.2 Causal Intervention Analysis

To understand slot specialization, we performed causal interventions on trained models (Figure 8):

- **Clamping**: Force 100% weight to a single slot
- **Disabling**: Zero out a slot and redistribute weight

Results show that performance is relatively robust to slot manipulation in diverse models, but collapsed models are highly sensitive.

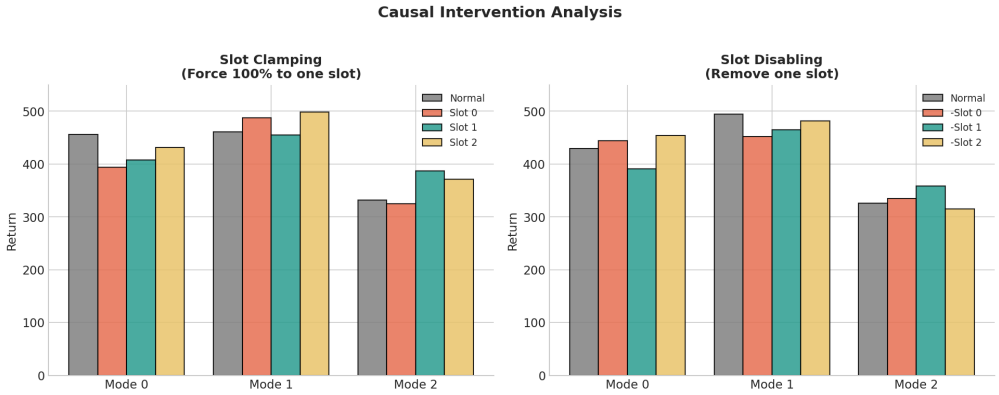

Figure 8: Causal intervention experiments showing effect of clamping (left) and disabling (right) individual slots.

A.3 SB3 PPO Fine-Tuning Baseline

Figure 9 shows that the SB3 PPO baseline can be successfully fine-tuned, achieving +77 average improvement. This demonstrates that adaptation *is* possible with full fine-tuning—but at the cost of forgetting.

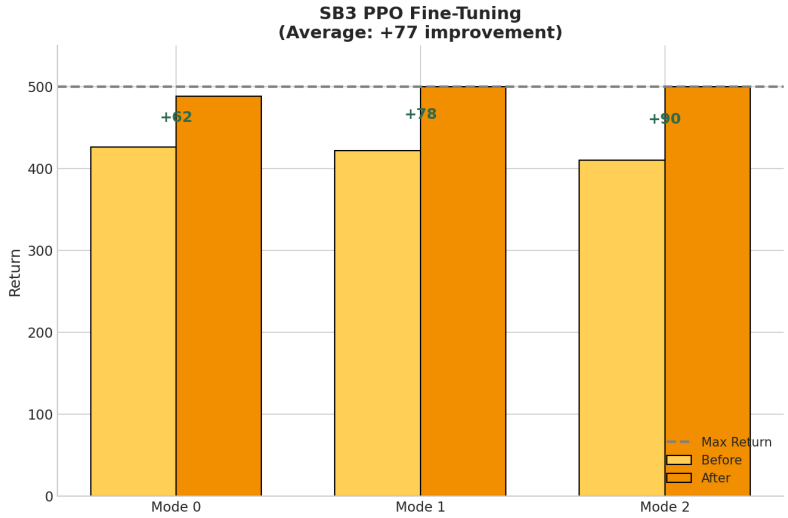

Figure 9: SB3 PPO fine-tuning achieves +77 average improvement, demonstrating that full adaptation is possible—but causes forgetting on non-adapted modes.

A.4 Computational Resources

All experiments were run on a single NVIDIA RTX 4090 GPU. Training times:

- 100k steps: ∼1 minute
- 300k steps: ∼3 minutes
- 500k steps: ∼5 minutes

Total compute for all 33+ models: approximately 3 GPU-hours.