Enhancing Reinforcement Learning for MuJoCo Tasks with Dynamic Memory in Tangled Program Graphs

Cyruss Allen Amante
McMaster University
Hamilton, Ontario, Canada
amantec@mcmaster.ca

Richard Li
McMaster University
Hamilton, Ontario, Canada
li1502@mcmaster.ca

Mark Angelo Cruz
McMaster University
Hamilton, Ontario, Canada
cruzm9@mcmaster.ca

Calvyn Siong
McMaster University
Hamilton, Ontario, Canada
siongc1@mcmaster.ca

Edward Gao
McMaster University
Hamilton, Ontario, Canada
gaoe2@mcmaster.ca

ABSTRACT

This paper investigates the impact of dynamic memory allocation within Tangled Program Graphs (TPG) for reinforcement learning in continuous control tasks, specifically within MuJoCo environments. TPG, an RL framework based on genetic programming, evolves agents composed of interconnected programs. We hypothesize that dynamic memory, which allows agents to adaptively adjust memory representation based on task demands, can enhance learning performance and efficiency compared to fixed-memory approaches. We explore this through single-task (STL) and multi-task (MTL) experiments on MuJoCo environments such as Inverted Pendulum, Half Cheetah, and Humanoid Standup. Our results demonstrate that dynamic memory leads to improved fitness scores and more effective program instruction utilization, particularly in multi-task scenarios, suggesting enhanced adaptability and knowledge sharing. We analyze the trade-offs between learning performance and computational efficiency, providing empirical validation for the theoretical benefits of dynamic memory in the genetic programming approach to RL like TPG.

KEYWORDS

Reinforcement Learning, Multi-Task Learning, Dynamic Memory, Tangled Program Graphs, Genetic Programming, MuJoCo, Continuous Control

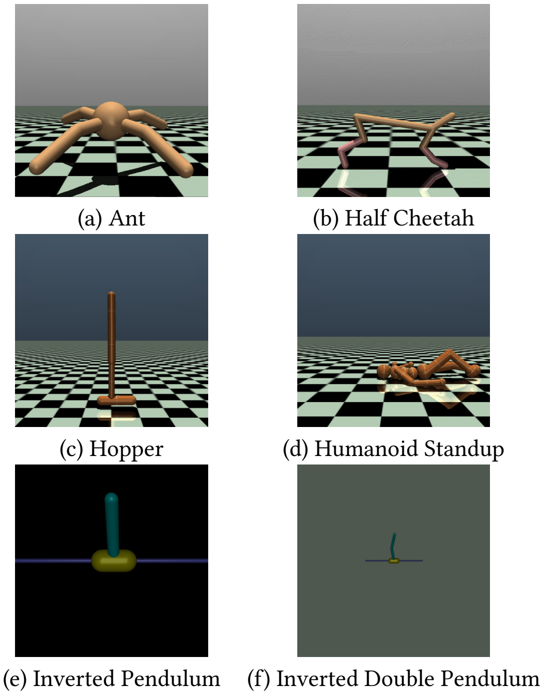
1 INTRODUCTION

Reinforcement learning (RL) has emerged as a powerful paradigm for training autonomous agents to perform complex tasks. A key challenge in RL is creating agents that can generalize to multiple tasks and environments, a problem known as multi-task learning (MTL). Real-world applications often require agents to adapt to diverse situations, making MTL a critical area of research.

Many existing reinforcement learning algorithms struggle with sample inefficiency, difficulty in handling continuous control, and poor generalization when applied to MuJoCo multi-task learning problems. Deep reinforcement learning methods, while powerful, often require vast amounts of training data and can be computationally expensive [4]. These methods often fail to capture the temporal dependencies and complex dynamics inherent in these environments, leading to sub-optimal performance, especially in partially observable scenarios.

1.1 MuJoCo

A significant domain for RL research, particularly in robotics, is physics simulation. MuJoCo (Multi-Joint dynamics with Contact) is a widely used physics engine, known for its accurate and efficient simulation of complex dynamics, contact forces, and articulated bodies [5]. Its ability to simulate realistic physics and provide diverse, challenging control tasks makes MuJoCo an invaluable tool for developing and evaluating reinforcement learning algorithms for robotics. The unique MTL and Single-Task Learning (STL) environments formulated in this work includes partially observable versions of the following 6 widely used RL benchmarks found on Gymnasium's MuJoCo suite [6]: Ant, Half Cheetah, Hopper, Humanoid Standup, Inverted Pendulum, and Inverted Double Pendulum, Figures 1(a) to 1(e).



(a) Ant
(b) Half Cheetah
(c) Hopper
(d) Humanoid Standup
(e) Inverted Pendulum
(f) Inverted Double Pendulum

Figure 1: MuJoCo Environments used in this work

1.2 Dynamic Memory

Dynamic memory plays a crucial role in evolving program graphs, particularly in multi-task learning (MTL), by providing a flexible and adaptive mechanism for encoding and processing temporal information. Unlike static memory architectures, which impose fixed storage structures, dynamic memory allows each program within an evolving graph to independently adjust its memory representation based on task demands. This adaptability facilitates more efficient learning and decision-making, ultimately accelerating evolutionary processes.

Dynamic memory within program graphs consists of three primary types:

- **Scalar Memory**: Stores single numerical values, useful for tracking individual state variables.
- **Vector Memory**: Represents data as structured arrays, allowing operations across multiple related values.
- **Matrix Memory**: Enables higher-dimensional representations, which can encode richer state information and facilitate more complex transformations.

Each program selects an appropriate memory type based on its computational needs, and mutations can modify both the type and dimensionality of the memory structures over the course of evolution.

1.3 Tangled Program Graphs

Tangled Program Graphs (TPG) is an RL framework developed by McMaster University's Creative Algorithm Lab under the guidance of Dr. Stephen Kelly. It leverages genetic programming principles to evolve agents capable of solving complex tasks in dynamic environments. Traditional RL methods, such as deep reinforcement learning (DRL), often rely on neural networks that require significant computational resources and large datasets for training. [1] In contrast, TPG uses genetic programming to evolve agents that can learn and adapt to their environment through a process of selection, mutation, and crossover. This unique approach allows TPG to achieve competitive performance in RL tasks with better computation efficiency than DRL methods. [1]

TPG revolves around the concept of emergent modularity, where agents are composed of interconnected programs that map environmental states to actions. These programs are organized into hierarchical structures, known as program graphs, that allow agents to break complex tasks into simpler subtasks. [3] To incorporate TPG with multi-task learning, an agent will have to be trained to perform multiple tasks sequentially. This is challenging as the agent must balance different objectives and environments while avoiding catastrophic forgetting (learning a new task causes agent to forget previously learned tasks, effectively wasting progress). [3] TPG's hierarchical and modular structure however, is able to tackle this challenge by allowing agents to dynamically adapt to different tasks by "recombining specialized behaviors". [3] TPG has been previously used to evolve agents capable of solving six distinct RL benchmarks from OpenAI's Classic Control suite, including CartPole, Acrobot, and Pendulum. [3] Currently, TPG is being further developed to gauge it's capability of integrating multi-task learning with more complex MuJoCo environments.

2 MOTIVATION

The primary motivation behind this research and development effort is to enhance the performance and efficiency of TPG in complex environments, specifically within the MuJoCo physics simulation framework. MuJoCo environments, such as Ant, Half Cheetah, and Humanoid Standup, present significant challenges due to their high-dimensional state and action spaces, as well as their dynamic nature. These environments require agents to learn proper control policies that generalize across diverse scenarios, making them ideal benchmarks for evaluating the scalability and adaptability of TPG.

The goal is to both improve performance and increase efficiency. Performance will be measured by the best fitness score achieved by the TPG agent, reflecting its ability to maximize cumulative rewards. Efficiency will be measured by the number of generations required for the agent to converge to an optimal state, which also reflects the time and computational resources needed for training. By integrating dynamic memory and optimizing the evolutionary process, this research aims to reduce the time to learn while maintaining or improving the quality of the learned policies.

3 RESEARCH QUESTIONS

The primary parameter given to the "`acmart`" document class is the *template style* which corresponds to the kind of publication or SIG publishing the work. This parameter is enclosed in square brackets and is a part of the `documentclass` command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the `acmsmall` template style:

- `acmsmall`: The default journal template style.
- `acmlarge`: Used by JOCCH and TAP.
- `acmtog`: Used by TOG.

The majority of conference proceedings documentation will use the `acmconf` template style.

- `sigconf`: The default proceedings template style.
- `sigchi`: Used for SIGCHI conference articles.
- `sigplan`: Used for SIGPLAN conference articles.

4 METHODOLOGY

This study evaluates task performance parameters such as generation time and fitness level through experiments conducted in their respective MuJoCo environments (see Figure 1). The methodology follows a two-phase approach: establishing baselines and integrating dynamic memory. We compare between baseline and dynamic memory-enhanced experiments that were conducted using statistical plots and TPG-generated data. All experiments utilized High Performance Parallel Compute (HPPC) resourcesprovided by the Digital Research Alliance of Canada ("The Alliance"). Each experiment was run with three random seeds for a three-hour duration.

4.1 Baseline Experiments

We conducted single-task and multi-task experiments using the following MuJoCo environments: inverted pendulum, inverted double pendulum, and half-cheetah. Single-task experiments used standardized hyperparameters listed in Table 1. Each experiment was assigned a specific memory_size parameter value basedon the dimensionality of the observation space, detailed in Table 2.

For multi-task experiments, we utilize the same MuJoCo environments. However, the root team size was increased to 3000, and *n_root_gen* was increased to 300 to accommodate the added complexity of multi-task learning. Two multi-task experiments were conducted:

1. **Two-environment multi-task:** Inverted pendulum and inverted double pendulum.
2. **Three-environment multi-task:** Inverted pendulum, inverted double pendulum, and half cheetah.

The initial mem_{size} parameter was set to 4 for the two-environment multi-task experiment and 17 for the three-environment multi-task experiment.

4.2 Dynamic Memory Experiments

To evaluate the benefits of adaptive memory allocation, we implemented a dynamic memory strategy by increasing the probability of changing memory size, p_{mem}, to 10% (0.1). The effects of this modification were assessed across the same baseline experiments.

- For single-task experiments, the minimum and maximum values for mem_{size} remained fixed at 2 and 32, respectively.
- For multi-task experiments, the minimum and maximum values for mem_{size} were dynamically adjusted based on the smallest and largest observation space dimensions among the participating environments.

4.3 Data Collection

For each experiment, performance metrics were systematically recorded and extracted from the `.std` output files, then parsed into structured `.csv` files. These `.csv` files were categorized into:

- **Timing Metrics:** Measures of computational timing.
- **Selection Metrics:** Data on operations used, fitness level, and instruction count.
- **Replacement Metrics:** Statistics on team and program numbers.
- **Removal Metrics:** Information on program and team deletions.

Key performance indicators analyzed include:

- **Best Fitness Score:** The highest fitness score achieved during execution.
- **Generations to Convergence:** The number of generations required to reach a specified performance threshold.
- **Effective Program Instructions:** The number of program instructions contributing to the final output.

After data collection, comparative analyses were conducted to evaluate the differences between baseline and dynamic memory configurations across both single-task and multi-task experiments. Visualization techniques, including comparative plots, were used to identify performance trends and assess the impact of dynamic memory integration.

5 BASELINE EXPERIMENTS RESULTS

This section presents the performance outcomes of the baseline experiments conducted in the MuJoCo environments, including single-task and multi-task scenarios. The evaluation metrics focus on **Best Fitness Score** and **Effective Program Instruction Count**, as illustrated in Figures 2 and 3, respectively.

5.1 Best Fitness Score Analysis

The **Best Fitness Score** metric, depicted in Figure 2, provides insights into the overall learning progress of the different experiments. The results highlight several key observations:

- **Single-task environments**: The fitness scores for Half Cheetah and Inverted Pendulum show that while both static and dynamic memory setups improve over time, dynamic memory consistently achieves higher fitness scores faster.
- **Multi-task environments**: Dynamic memory demonstrates a clear advantage, particularly in the Two- and Three-task multi-task setups. It outperforms static memory across generations, highlighting its ability to adapt more effectively to increasing task complexity.

5.2 Program Instruction Count Analysis

Figure 3 presents the results of the **Effective Program Instruction Count**, a key metric reflecting the number of active program instructions contributing to task performance.

- **Single-task environments**: Dynamic environments show more fluctuations in active instruction counts, indicating frequent adaptation for optimization. In contrast, static environments stabilize early, limiting flexibility.
- **Multi-task environments**: The Effective Program Instruction Count is significantly higher in dynamic multi-task setups, suggesting better management of complex learning structures. Static environments struggle to scale efficiently.

6 RESULTS AND CONCLUSION

This research investigated the impact of dynamic memory within Tangled Program Graphs (TPG) for reinforcement learning in MuJoCo environments, addressing key questions regarding its role in STL, MTL, computational efficiency, and validation of theoretical benefits.

6.1 RQ1: Role in Single-Task Learning

Our experiments in single-task MuJoCo control tasks (Half Cheetah, Inverted Pendulum) demonstrate that TPG agents with dynamic memory achieve higher fitness scores and faster learning convergence compared to those with fixed memory. This suggests that dynamic memory enhances the agent's ability to adapt to task-specific dynamics even in relatively simple, fully observable environments.

6.2 RQ2: Role in Multi-Task Learning & Adaptability

In multi-task learning scenarios (Two-Environment and Three-Environment setups), dynamic memory exhibited a clear advantage. Agents with dynamic memory showed significantly improved adaptability and performance, achieving higher fitness scores across generations compared to their fixed-memory counterparts. This indicates that dynamic memory facilitates better task identification,

Table 1: Hyperparameters for MuJoCo environments, single-task team population, and program population

MuJoCo parameters		Team population		Program population	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Max timestep	1000	Agent (root team) population size	1000	Initial program size	10
Reward control weight	0.5	Initial team size	1	p_{delete}	0.2
Number of training evaluations	20	Max team size	10	$p_{\text{add}}, p_{\text{swap}}, p_{\text{mutate}}$	0.25
Number of test evaluations	1	n_root_gen	100	mem_{min}	2
Number of validation evaluations	0			mem_{max}	32
				p_{mem}	0.0

Note: n_root_gen **denotes the number of new root teams to create each generation.** $p_{\mathbf{x}}$ **in which** $x \in \{add, delete, swap, mutate\}$ **are the probabilities of adding, deleting, swapping, or mutating instructions within a program.** $p_{\mathbf{mem}}$ **is the probability of changing the memory size,** $mem_{\mathbf{size}}$**, within the** $mem_{\mathbf{min}}$ **and** $mem_{\mathbf{max}}$ **interval.**

Table 2: Observation and action space sizes for the considered problems [2]

Environment	Obs. $\mathcal{O}$	Act. $\mathcal{A}$
Inverted pendulum	$\mathbb{R}^4$	[-3, -3]
Inverted double pendulum	$\mathbb{R}^9$	[-1, 1]
Half cheetah	$\mathbb{R}^{17}$	[-1, 1]

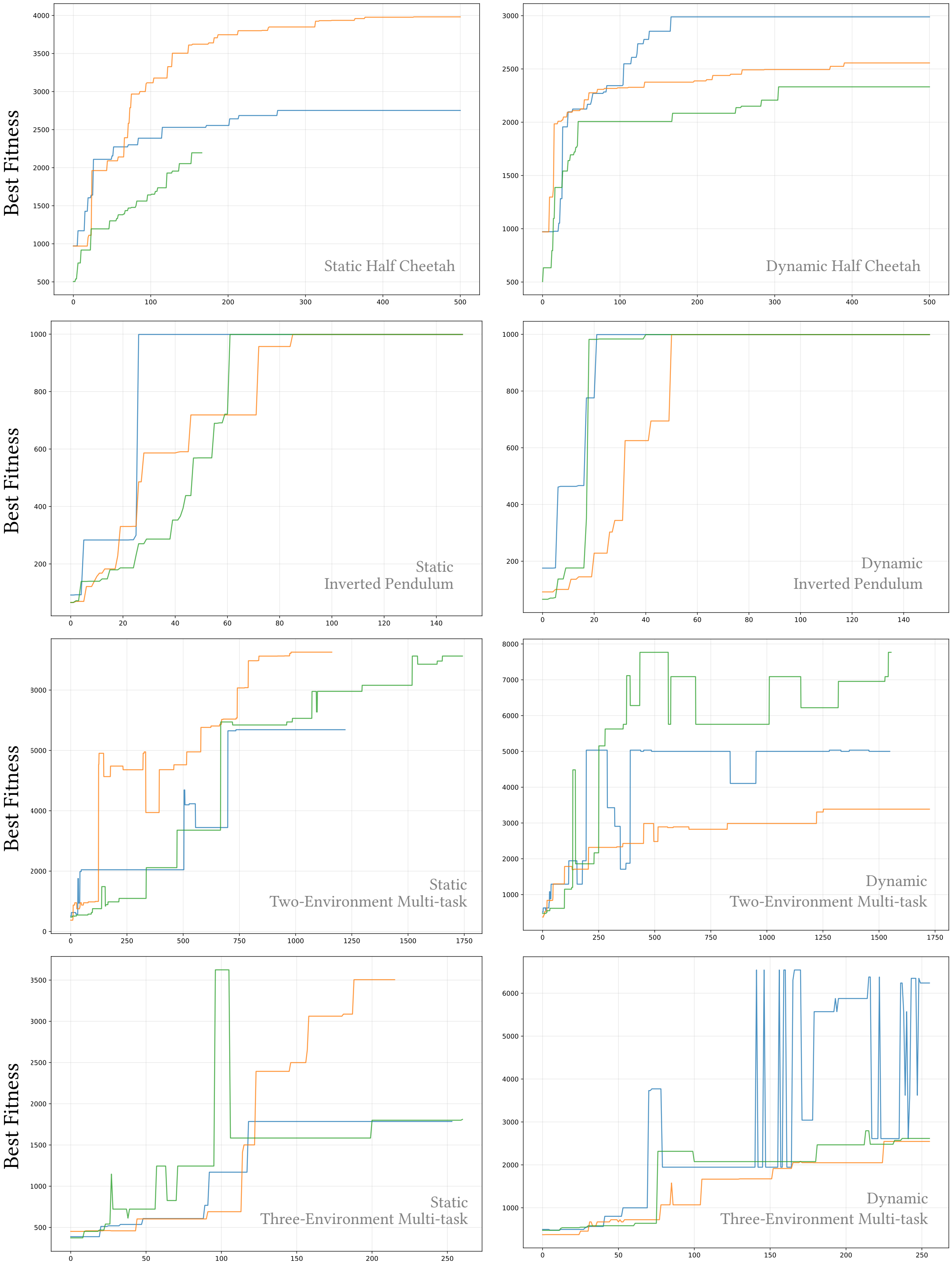


Figure 2: Best Fitness Score results from Single and Multi-task Baseline Experiments

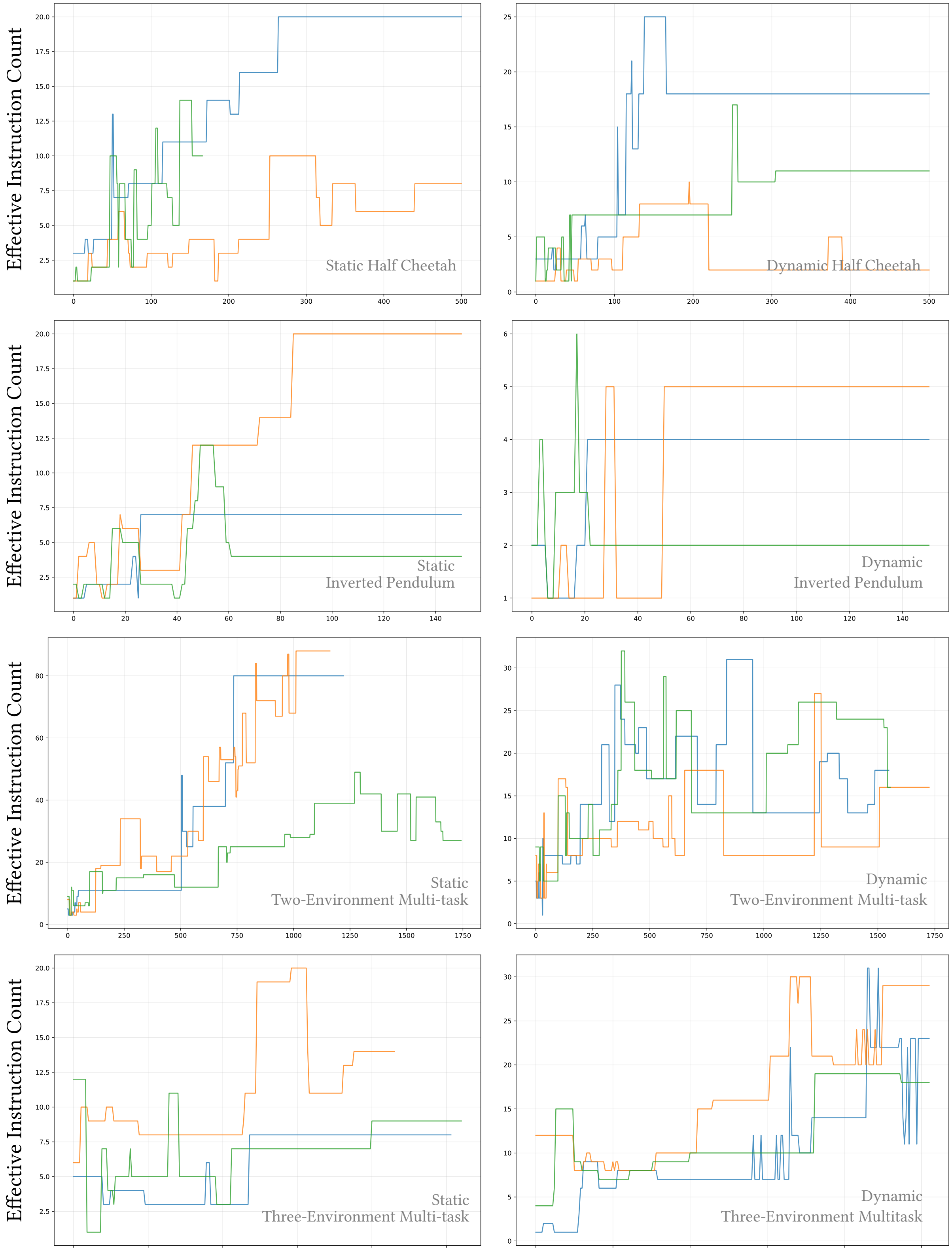


Figure 3: Effective Program Instruction Count results from Single and Multi-task Baseline Experiments

knowledge sharing, and policy adaptation across multiple tasks with varying state dynamics.

6.3 RQ3: Performance vs. Computational Efficiency

The results indicate that the performance gains achieved through dynamic memory do not come at a prohibitive computational cost. While dynamic memory agents demonstrate more fluctuations in active program instruction counts, indicating frequent adaptation, the overall learning efficiency, as measured by generations to convergence and best fitness score, is enhanced. This suggests that dynamic memory allows for more effective utilization of computational resources, leading to improved learning outcomes without significant overhead.

6.4 RQ4: Validation of Theoretical Benefits

Our empirical findings largely validate the theoretical motivations for integrating dynamic memory into TPG. The improved performance in both single-task and multi-task scenarios supports the hypothesis that dynamic memory enables more effective temporal credit assignment and hierarchical problem decomposition. The dynamic memory agents exhibited a more flexible reallocation of memory resources, aligning with the problem's demands and demonstrating the anticipated benefits of better handling partial observability and long-term dependencies.

In conclusion, this research provides strong evidence for the benefits of dynamic memory within TPG for reinforcement learning in complex MuJoCo environments. The ability to adaptively allocate memory based on task demands enhances both learning performance and efficiency, particularly in multi-task scenarios. These findings highlight the potential of dynamic memory as a key component in developing more robust and adaptable RL agents capable of tackling real-world challenges. Future work should focus on further optimizing the dynamic memory allocation process and exploring its application in even more complex and partially observable environments.

ACKNOWLEDGMENTS

To Dr. Stephen Kelly, for guiding us throughout the research process and providing valuable insights and feedback.

REFERENCES

[1] Tanya Djavaherpour, Ali Naqvi, Eddie Zhuang, and Stephen Kelly. 2025. Evolving Many-Model Agents with Vector and Matrix Operations in Tangled Program Graphs. In *Genetic Programming Theory and Practice XXI*, Stephan M. Winkler, Wolfgang Banzhaf, Ting Hu, and Alexander Lalejini (Eds.). Springer Nature Singapore, Singapore, 87–105. doi:10.1007/978-981-96-0077-9_5

[2] Farama Foundation. 2024. Gymnasium MuJoCo Environments. https://gymnasium.farama.org/environments/mujoco/

[3] Stephen Kelly, Tatiana Voegerl, Wolfgang Banzhaf, and Cedric Gondro. 2021. Evolving Hierarchical Memory-Prediction Machines in Multi-Task Reinforcement Learning. arXiv:2106.12659 [cs.NE] https://arxiv.org/abs/2106.12659

[4] Volodymyr Mnih and Koray Kavukcuoglu. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. doi:10.1038/nature14236

[5] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* 1, 1 (2012), 5026–5033. https://api.semanticscholar.org/CorpusID:5230692

[6] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv:2407.17032 [cs.LG] https://arxiv.org/abs/2407.17032

Received 16 March 2025