

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

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

Mark Angelo Cruz  
McMaster University  
Hamilton, Ontario, Canada  
cruz9@mcmaster.ca

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

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

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

## ABSTRACT

This paper investigates the impact of dynamic memory allocation within Tangled Program Graphs (TPG) for reinforcement learning in continuous control tasks. 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 Inverted Double Pendulum. Our results demonstrate a complex relationship between dynamic memory and performance. While some single-task and multi-task scenarios showed improvements in fitness scores or computational efficiency with dynamic memory, others exhibited no significant change or even a decrease in performance. These findings suggest that the benefits of dynamic memory within TPG are task-dependent and may not generalize across all environments.

## 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 through trial-and-error interaction with an environment. However, real-world applicability demands that agents not only achieve high performance but also adapt efficiently to varying task demands and environmental dynamics. This paper focuses on enhancing the TPG framework, a genetic programming-based RL method, for continuous control tasks in the MuJoCo physics simulation environment.

Traditional deep reinforcement learning (DRL) methods, while powerful, often require substantial computational resources and large datasets for effective training [6]. In contrast, TPG leverages emergent modularity and an adaptive, dynamically evolving program structure, offering a potentially more computationally efficient alternative [5].

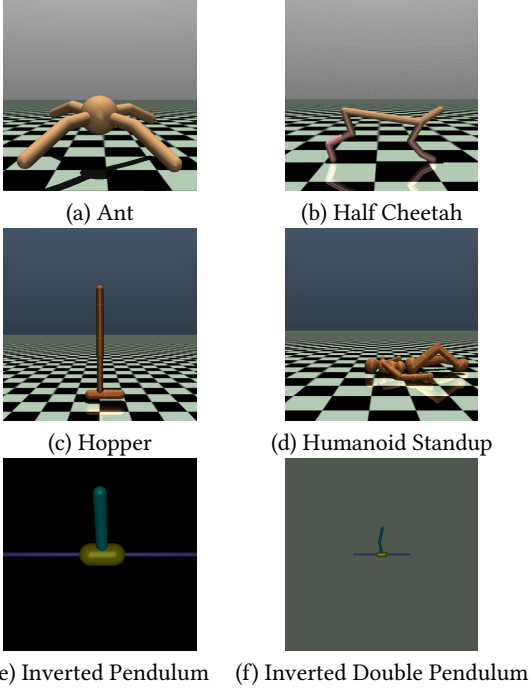
Our key contribution lies in investigating the impact of dynamic memory allocation within TPG. We hypothesized that allowing TPG agents to adaptively adjust their memory representation based on task demands would improve both learning performance and sample efficiency in complex MuJoCo environments. We tested this hypothesis through a series of single-task and multi-task experiments, analyzing the trade-offs between performance, computational cost, and effective program structure, and detail our methodology and experimental setup in Section 4. We will go on to detail the data collection and statistical comparison in Section 5, and lastly provides discussion and conclusions of our findings in Section 6. Our findings reveal a more complex relationship than initially anticipated, with dynamic memory demonstrating benefits in some scenarios but not consistently across all tasks and environments. We discuss the implications of these results for the design and application of dynamic memory in genetic programming-based RL.

### 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 [7]. Its ability to simulate realistic physics and provide diverse, challenging multi-action continuous 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 the following 6 widely used RL benchmarks found on Gymnasium’s MuJoCo suite [8]: Ant, Half Cheetah, Hopper, Humanoid Standup, Inverted Pendulum, and Inverted Double Pendulum, Figures 1(a) to 1(e).

### 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



**Figure 1: MuJoCo Environments used in this work**

learning and decision-making, ultimately accelerating evolutionary processes.

Dynamic memory within program graphs consists of two primary types:

- **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 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. [2] 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. [2]

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. [5] 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). [5] 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”. [5] TPG has been previously used to evolve agents capable of solving six distinct RL benchmarks from OpenAI’s Classic Control suite, including Cart-Pole, Acrobot, and Pendulum. [5] Currently, TPG is being further developed to gauge its 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. Unlike the previous environments that were explored with TPG such as the Atari video game testbed or OpenAI’s Classic Control suite, MuJoCo environments are much more similar to real-world, complex, and dynamic robotics problems. This makes them a critical step forward in testing TPG’s applicability to real-world applications, such as autonomous robotics and embedded systems.

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

Building on the motivation that memory mechanisms can significantly improve an agent’s ability to handle temporal dependencies in reinforcement learning (RL) [1], we investigate the specific contribution of **dynamic memory** within the Tangled Program Graphs (TPG) framework. Prior work has shown that TPG agents without any persistent memory perform poorly on tasks requiring temporal integration (e.g., partially observable environments)[5], underscoring the need for effective memory strategies. While the standard TPG uses a fixed-size (static) memory for all tasks, a *dynamic* memory approach—where the agent can allocate and manage memory

on the fly—offers a theoretically more flexible alternative [4]. Such dynamic memory could allow hierarchical RL agents to encode and retrieve information more adaptively over time [4]. We therefore pose the following research questions, which align with our methodology and experiments on continuous control tasks (MuJoCo environments):

- (1) **RQ1: Role in Single-Task Learning** — *How does the incorporation of dynamic memory influence learning performance in single-task MuJoCo control tasks using TPG, compared to a fixed-memory setup?* This question examines whether a TPG agent endowed with dynamic memory attains higher rewards or learns more efficiently on individual tasks (each trained in isolation) versus the baseline of fixed (static) memory. We seek to determine if dynamic memory provides any tangible benefit in single-task scenarios, or if fully observable single tasks already suffice with a static memory mechanism.
- (2) **RQ2: Role in Multi-Task Learning & Adaptability** — *Does dynamic memory improve an agent’s adaptability and performance in multi-task learning scenarios within TPG?* In other words, when a single TPG agent must handle **multiple** MuJoCo tasks, does a dynamic memory mechanism enable better task identification, knowledge sharing, or policy adaptation across tasks compared to fixed memory? This question probes whether dynamic memory helps the hierarchical TPG agent adjust to different tasks (potentially with diverse state dynamics) more effectively. Prior studies suggest that memory is crucial for multi-task agents to approach the performance of task-specific agents [5], so here we ask if allowing memory to be managed dynamically further enhances the agent’s multi-task learning capability and resilience to task variations.
- (3) **RQ3: Performance vs. Computational Efficiency** — *What are the effects of using dynamic memory on learning efficiency and computational cost in TPG, relative to a fixed-memory approach?* We compare dynamic and fixed memory configurations to assess trade-offs between **learning performance** (e.g., speed of convergence, final reward achieved) and **computational efficiency** (e.g., runtime or memory overhead). This question addresses whether any gains from dynamic memory come at the expense of significantly higher computational complexity. Ideally, an enhanced memory system should improve learning outcomes without incurring prohibitive overhead. For instance, recent work demonstrated that better memory management can boost performance while **preserving** computational efficiency [1]. Here, we examine if a dynamic memory design achieves a similar balance in practice.
- (4) **RQ4: Validation of Theoretical Benefits** — *Do our empirical findings validate the theoretical motivations for integrating dynamic memory into a hierarchical RL framework like TPG?* Dynamic memory is hypothesized to enable more effective temporal credit assignment and hierarchical problem decomposition [4]. This question asks whether the introduction of dynamic memory in TPG indeed yields these anticipated benefits. Specifically, we investigate if the dynamic memory agent exhibits improved **temporal integration** of past

information and a more flexible reallocation of memory resources that align with the problem’s demands. By comparing the behaviors and structures emerging in dynamic-memory TPG agents versus static-memory ones, we aim to confirm whether dynamic memory provides the expected advantages (such as better handling of partial observability and long-term dependencies) in a real experimental setting.

## 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) resources provided 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 based on 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,  $p_{\text{mem}}$ , 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.

**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     | $p_{\text{delete}}$                                  | 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     | $n_{\text{root\_gen}}$            | 100   | $mem_{\text{min}}$                                   | 2     |
| Number of validation evaluations | 0     |                                   |       | $mem_{\text{max}}$                                   | 32    |
|                                  |       |                                   |       | $p_{\text{mem}}$                                     | 0.0   |

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

**Table 2: Observation and action space sizes for the considered problems [3]**

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

### 4.3 Data Collection

For each experiment, performance metrics were systematically recorded and outputted 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.

## 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**, **Effective Program Instruction Count** and **Runtime Complexity**, as illustrated in the collection of figures in Appendix A.

### 5.1 Best Fitness Score Analysis

The **Best Fitness Score** metric, depicted in Figure 2 for single-task experiments and Figures 4 and 6 for multi-task experiments, provides insights into the overall learning progress of the different experiments. The results highlight several key observations:

- **Single-task environments:** Inverted Pendulum and Inverted Double Pendulum show improvement in fitness scores over time for dynamic memory setups, displaying consistently achieving higher fitness scores more quickly. In contrast, Half Cheetah performs worse with dynamic memory, as indicated by its lower fitness scores.
- **Multi-task environments:** Dynamic memory does not show a clear advantage in best fitness score, particularly in both two- and three-environment multi-task setups. In the two-environment experiments, static memory outperforms dynamic memory by consistently achieving higher fitness scores across both environments. However, in the three-environment experiments, dynamic memory surpasses static memory, particularly in Half Cheetah and Inverted Pendulum.

### 5.2 Runtime Analysis

Figure 3 presents the results of the **Effective Program Instruction Count** for single-task experiments, a key metric reflecting the number of active program instructions contributing to task performance. Figures 5 and 7 present the results of the **Runtime Complexity** for multi-task experiments, measuring the computational cost of executing the program under different conditions.

- **Single-task environments:** The effective program instruction count is significantly lower in dynamic memory setups for Inverted Pendulum and Inverted Double Pendulum, indicating more efficient program execution. In Half Cheetah, the count remains unchanged between static and dynamic memory.
- **Multi-task environments:** In two-environment experiments, runtime complexity is significantly lower with dynamic memory, showing a consistent reduction in execution

time across generations and improved computational efficiency. However, in three-environment experiments, runtime complexity remains unchanged, suggesting that dynamic memory does not provide a significant advantage in this setting.

## 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 single-task learning (STL), multi-task learning (MTL), computational efficiency, and validation of theoretical benefits.

### 6.1 RQ1: Role in Single-Task Learning

Our experiments in single-task MuJoCo control tasks revealed mixed results. While Inverted Pendulum and Inverted Double Pendulum showed improved fitness scores and faster learning convergence with dynamic memory, Half Cheetah performed worse compared to fixed memory. This suggests that the benefits of dynamic memory in single-task scenarios are task-dependent, and may not generalize across all environments.

### 6.2 RQ2: Role in Multi-Task Learning & Adaptability

In multi-task learning scenarios, the impact of dynamic memory was also nuanced. In the two-environment (Inverted Pendulum, Inverted Double Pendulum) setup, static memory outperformed dynamic memory in terms of best fitness score. However, in the three-environment (Inverted Pendulum, Inverted Double Pendulum, Half Cheetah) setup, dynamic memory showed improved performance, particularly in Half Cheetah and Inverted Pendulum. These findings indicate that the effectiveness of dynamic memory in multi-task learning may depend on the complexity and diversity of the task set.

### 6.3 RQ3: Performance vs. Computational Efficiency

The analysis of computational efficiency revealed that dynamic memory can lead to reduced runtime complexity in certain scenarios. In single-task environments, Inverted Pendulum and Inverted Double Pendulum exhibited lower effective program instruction counts with dynamic memory. Moreover, the two-environment multi-task setup showed significantly lower runtime complexity with dynamic memory. However, in the three-environment multi-task setup, runtime complexity remained largely unchanged. This suggests that dynamic memory can improve computational efficiency in some cases, but the benefits are not consistent across all environments and task configurations.

### 6.4 RQ4: Validation of Theoretical Benefits

The empirical findings provide mixed support for the theoretical benefits of dynamic memory. While dynamic memory demonstrated potential for improved adaptability and computational efficiency in specific scenarios, the lack of consistent performance gains across all tasks and environments suggests that the theoretical advantages

may not always translate directly into practical improvements. Factors such as task complexity, environment characteristics, and the specific implementation of dynamic memory within TPG can all influence its effectiveness.

In conclusion, this research demonstrates that the impact of dynamic memory within TPG for reinforcement learning in MuJoCo environments is complex and task-dependent. While dynamic memory shows promise for improving performance and efficiency in certain scenarios, its benefits are not universal and can be influenced by various factors. Further research is needed to better understand the conditions under which dynamic memory is most effective and to develop strategies for optimizing its application in diverse and challenging RL 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, and Stephen Kelly. 2024. Tangled Program Graphs with Indexed Memory in Control Tasks with Short Time Dependencies. In *Proceedings of the 16th International Joint Conference on Computational Intelligence (IJCCI 2024)*. SCITEPRESS, 296–303. doi:10.5220/0013016800003837
- [2] 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
- [3] Farama Foundation. 2024. Gymnasium MuJoCo Environments. <https://gymnasium.farama.org/environments/mujoco/>
- [4] Stephen Kelly, Robert J. Smith, Malcolm I. Heywood, and Wolfgang Banzhaf. 2021. Emergent Tangled Program Graphs in Partially Observable Recursive Forecasting and ViZDoom Navigation Tasks. *ACM Transactions on Evolutionary Learning and Optimization* 1, 3 (2021), 1–41. doi:10.1145/3468857
- [5] 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>
- [6] Volodymyr Mnih and Koray Kavukcuoglu. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. doi:10.1038/nature14236
- [7] 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>
- [8] 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>

## A ADDITIONAL FIGURES

### A.1 Single Tasks - Best Fitness Score Results

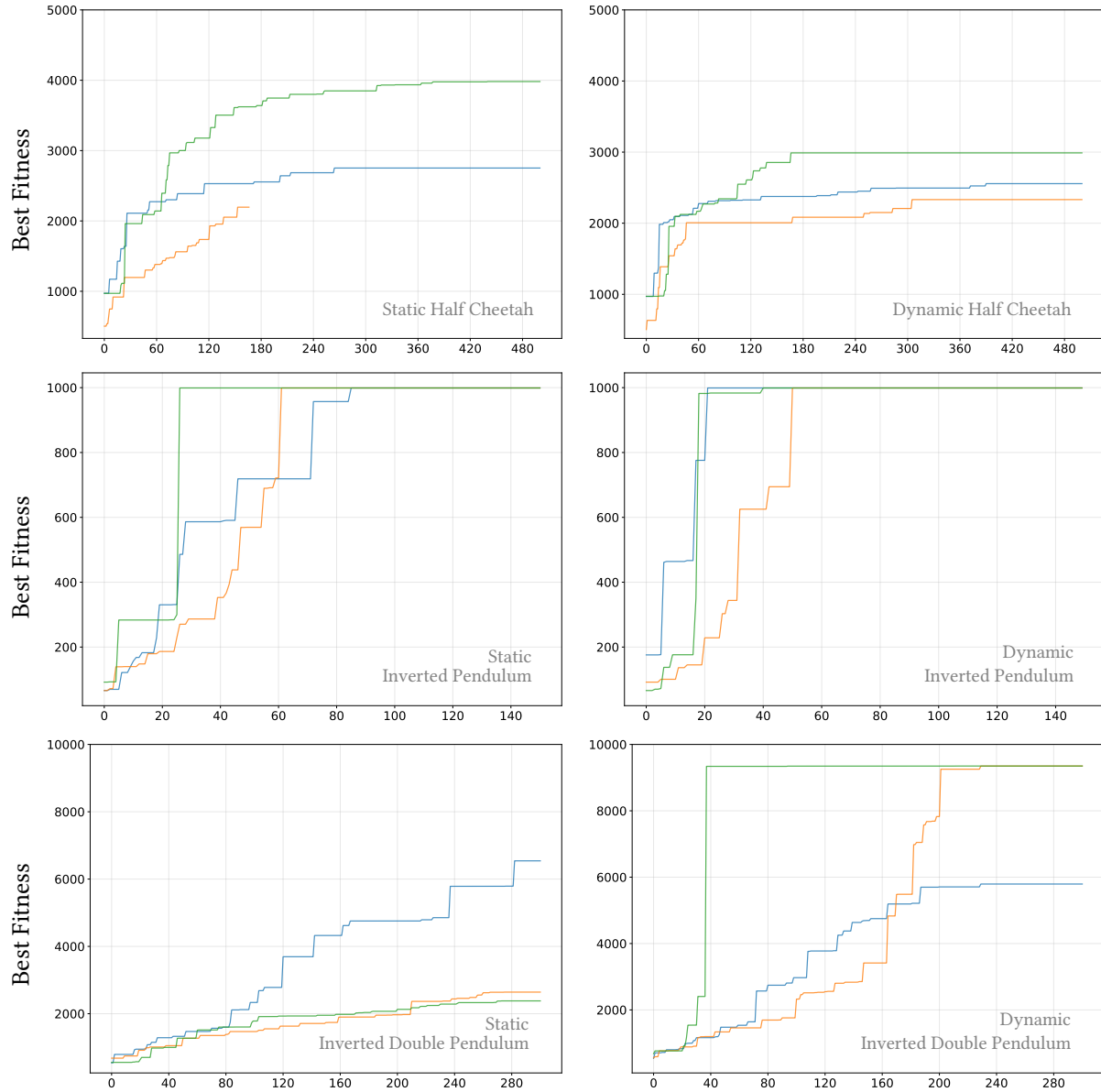


Figure 2: Best Fitness Score results from Single-task Baseline Experiments

## A.2 Single Tasks - Effective Program Instruction Count Results

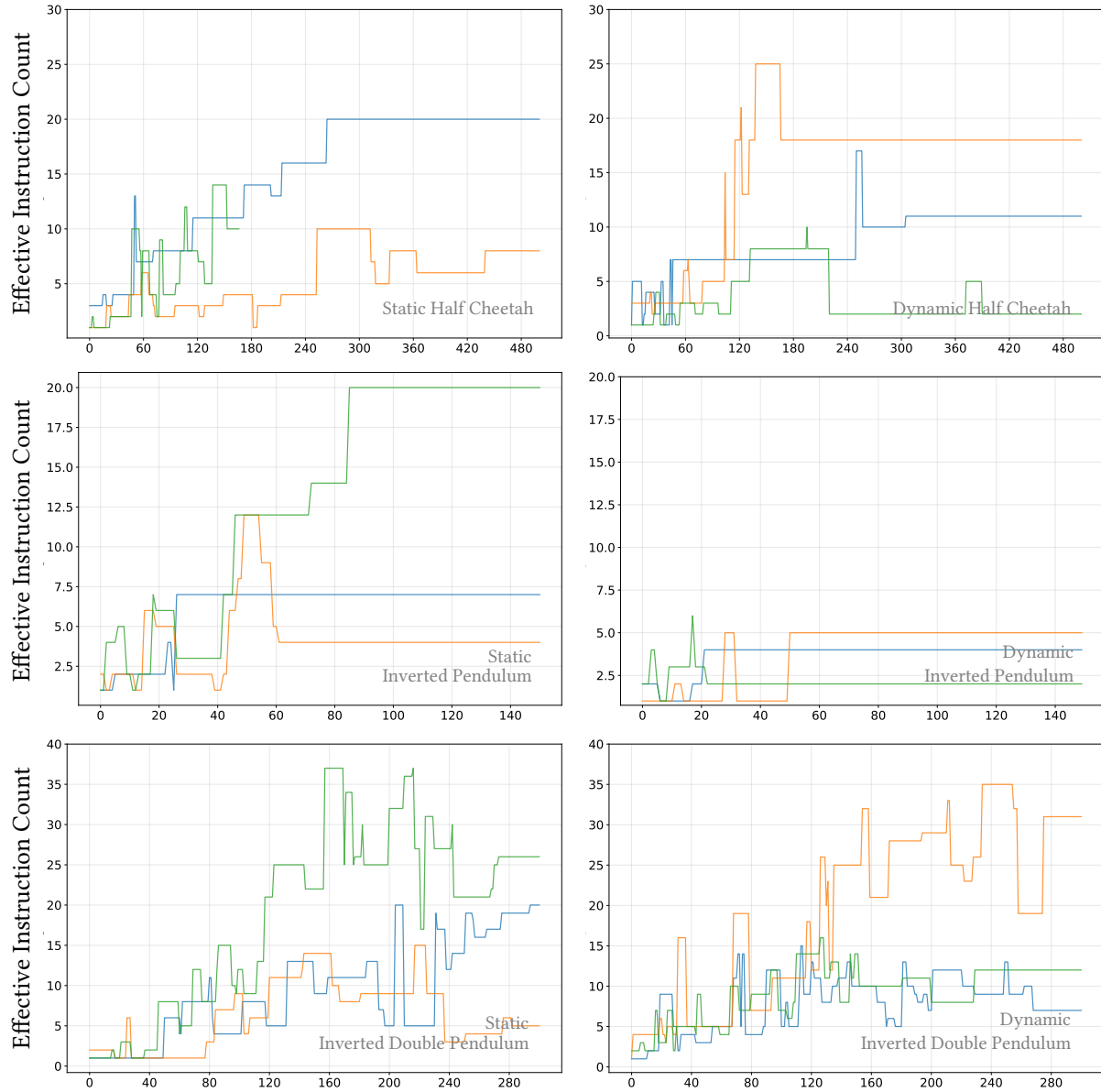


Figure 3: Effective Program Instruction Count results from Single-task Baseline Experiments

### A.3 Two-Experiment Multi-task - Best Fitness Score Results

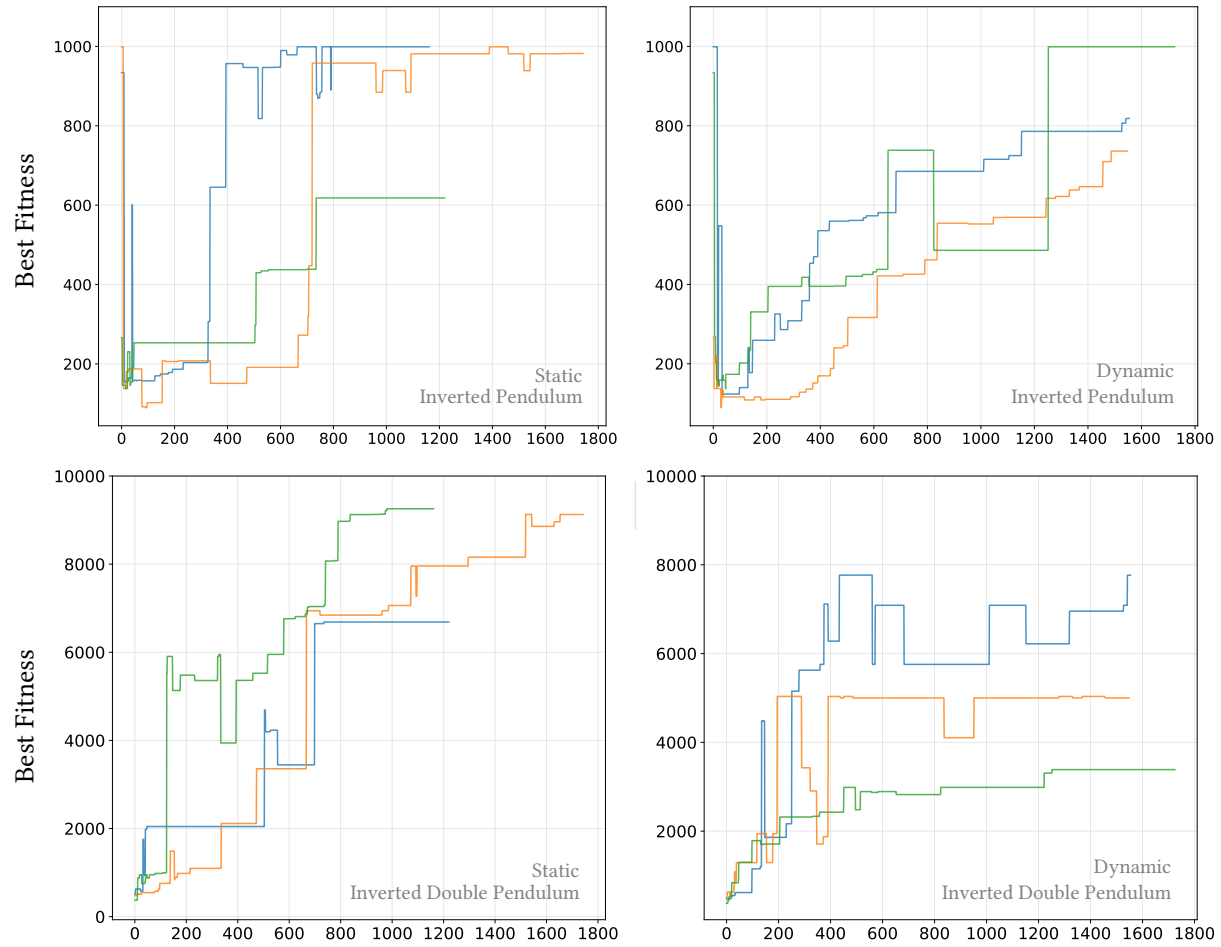


Figure 4: Best Fitness Score results from Two-environment Multi-task Baseline Experiments



#### A.4 Two-Experiment Multi-task - Runtime Complexity Results

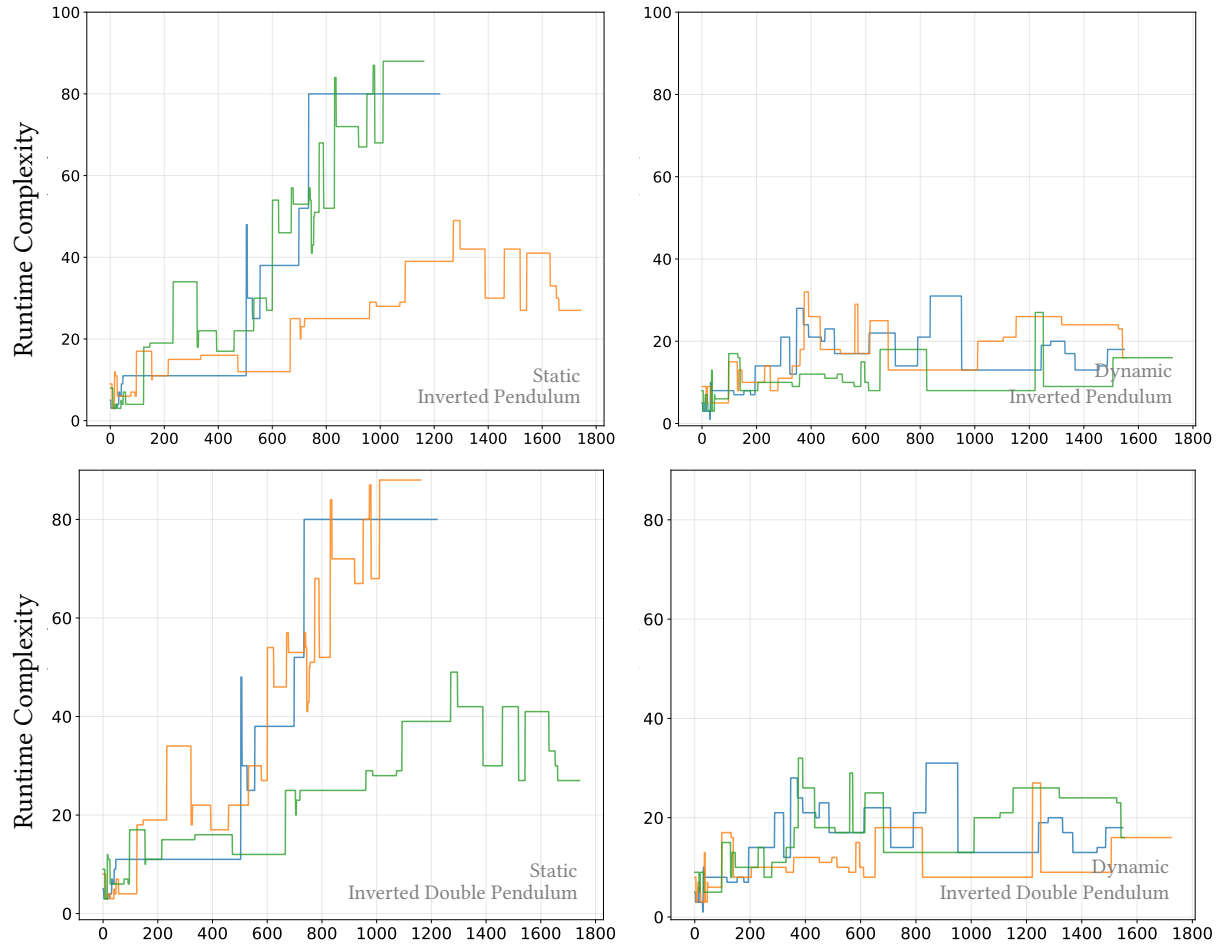


Figure 5: Runtime Complexity results from Two-environment Multi-task Baseline Experiments

### A.5 Three-Experiment Multi-task - Best Fitness Score Results

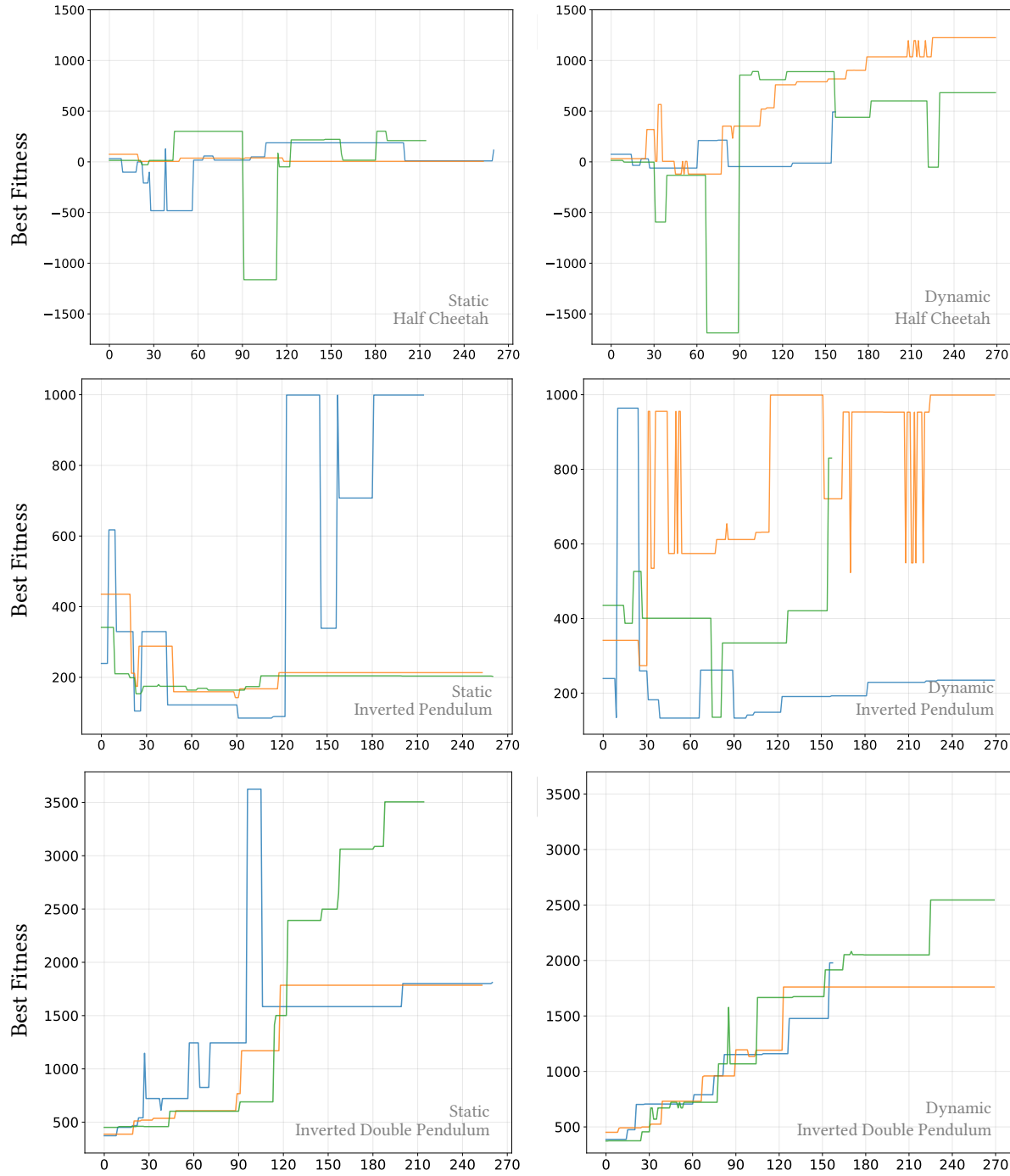


Figure 6: Best Fitness Score results from Three-environment Multi-task Baseline Experiments

### A.6 Three-Experiment Multi-task - Runtime Complexity Results

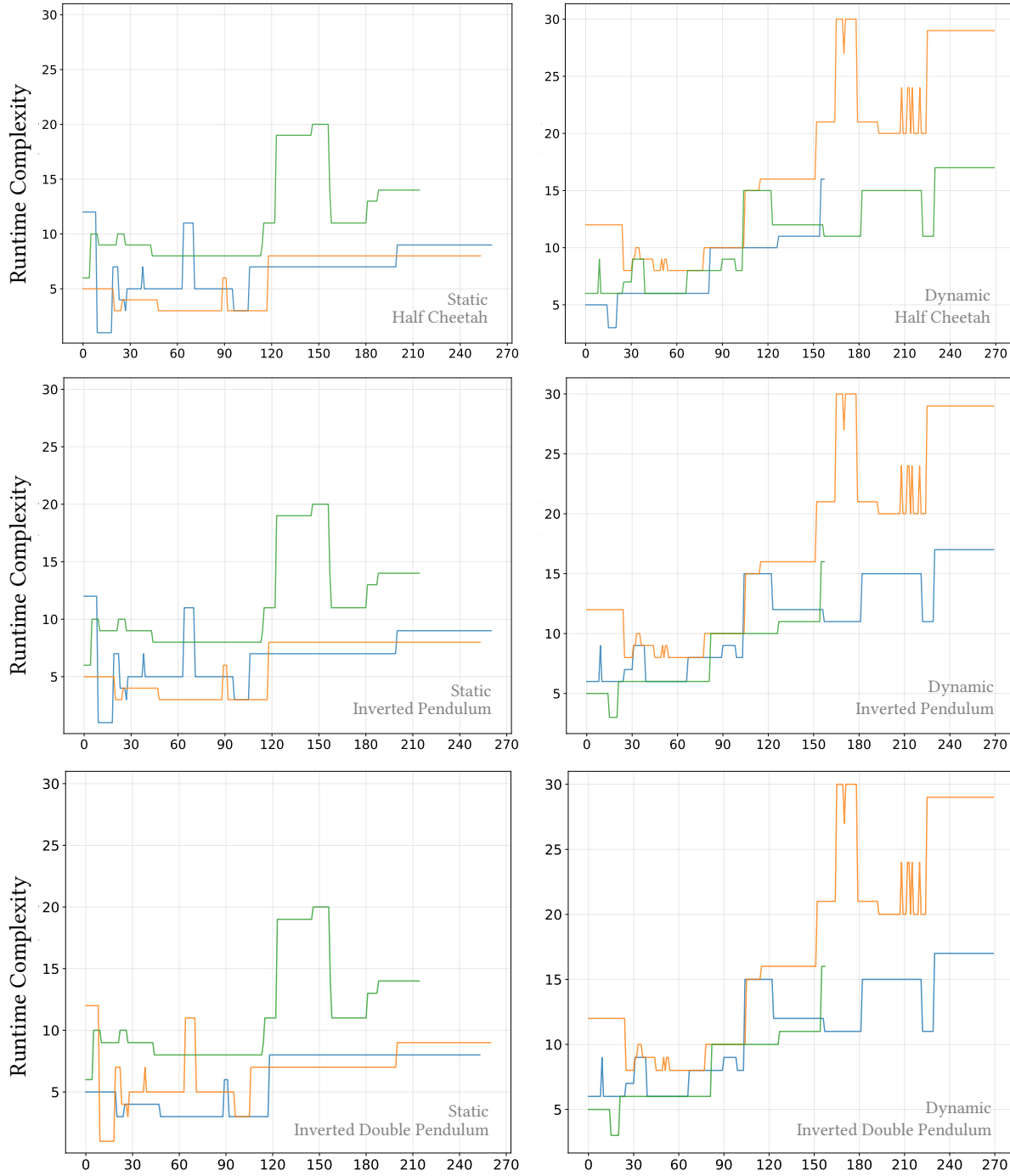


Figure 7: Runtime Complexity results from Three-environment Multi-task Baseline Experiments