Enhancing Deep Reinforcement Learning: Methods and Applications for Real-world Challenges

Jikun Kang

School of Computer Science

McGill University, Montreal

August, 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

Abstract

Despite impressive achievements in domains such as video games, the ancient strategy game of Go, and high-dimensional control problems, the application of Deep Reinforcement Learning (DRL) in real-world scenarios remains challenging due to issues of data efficiency, training complexity, model generalization, and safety concerns. Addressing these challenges, the thesis introduces several innovative methods. Firstly, we propose a multi-teacher knowledge distillation approach for model-based RL, aimed at improving data efficiency. Secondly, a unified automatic curriculum learning framework is presented to alleviate training complexity. Thirdly, we introduce an internal working memory module, aimed at enhancing model generalization. Lastly, we address safety concerns via a Hierarchical Policy Learning (HPL) framework, coordinating actions between different policy levels to avoid potential conflicts and ensure safe operations. Our methods have been applied and evaluated on a range of tasks, including network load-balancing, robotic manipulation, Atari games, and object manipulation tasks. The results demonstrate the effectiveness of the proposed approaches in improving the performance of DRL algorithms

in real-world applications, signifying their potential in tackling the challenges of implementing DRL in complex, dynamic, and safety-critical domains.

Abrégé

Malgré des réalisations impressionnantes dans des domaines tels que les jeux vidéo, l'ancien jeu de stratégie Go, et des problèmes de contrôle à haute dimension, l'application de la DRL dans des scénarios du monde réel reste un défi en raison de problèmes d'efficacité des données, de complexité de l'entraînement, de généralisation du modèle et de préoccupations de sécurité. Pour répondre à ces défis, cette thèse introduit plusieurs méthodes innovantes. Tout d'abord, nous proposons une approche de distillation de connaissances multi-enseignants pour le RL basé sur le modèle, visant à améliorer l'efficacité des données. Ensuite, un cadre d'apprentissage de curriculum automatique unifié est présenté pour alléger la complexité de l'entraînement. En troisième lieu, nous introduisons un module de mémoire de travail interne, visant à améliorer la généralisation du modèle. Enfin, nous abordons les préoccupations de sécurité via un cadre d'apprentissage de politique hiérarchique (HPL), coordonnant les actions entre différents niveaux de politique pour éviter les conflits potentiels et assurer des opérations sûres. Nos méthodes ont été appliquées et évaluées sur une gamme de tâches, y compris l'équilibrage

de charge de réseau, la manipulation robotique, les jeux Atari, et les tâches de manipulation d'objets. Les résultats démontrent l'efficacité des approches proposées pour améliorer la performance des algorithmes de DRL dans les applications du monde réel, signifiant leur potentiel pour relever les défis de la mise en œuvre de DRL dans des domaines complexes, dynamiques et critiques pour la sécurité.

Acknowledgements

As I pen down this thesis, my heart throbs with a mixture of exhilaration and gratitude. It's hard to believe that the boy who embarked on this journey, brimming with uncertainty and dreams, has finally reached the finish line of his Ph.D. journey. The past six years, akin to a fleeting glance, feel like a vivid movie reel in my mind, a montage of joy, sorrow, and everything in between.

While I've often told my friends that I might have reconsidered pursuing a doctorate if given a chance to start over, I hold no regrets. The commitment I've made, the wisdom I've gained, and the adversity I've overcome have been instrumental in shaping my journey. I've aspired to be a scientist since childhood and find a profound joy in research. It gives me a sense of purpose and contributes to society in a way that ensures my life is not devoid of meaning.

This milestone wouldn't have been possible without the unwavering support and encouragement of my supervisors, collaborators, labmates, friends, and family. Their love, kindness, and inspiring words turned my most difficult days into rewarding experiences,

and for this, I am eternally grateful.

At the forefront of those to whom I am indebted is my supervisor, Professor Xue (Steve) Liu. His unwavering support, invaluable guidance, and ceaseless encouragement were vital in the completion of my Ph.D. Professor Liu's visionary thinking and profound wisdom have steered me towards becoming a skilled researcher and scientist. His enthusiasm for research, teaching, and student supervision will forever be a beacon of inspiration as I steer my future career. The opportunity to work on my doctoral dissertation under his guidance, and benefiting from his wealth of research experience, has been an honor beyond words.

I extend my gratitude to my collaborators, including Professor Jie Fu, Professor Gregory Dedek, Professor Christopher Pal, Dr. Xi Chen, Dr. Ju Wang, Dr. Miao Liu, Dr. Xingdi Yuan, Dr. Di Wu, Dr. Romain Laroche, Dr. Adam Trischler, and many more. Their insightful critiques and recommendations have been invaluable in this thesis and my other research work. It's been a privilege to collaborate with them. In particular, I'm immensely grateful to Professor Jie Fu, a wellspring of knowledge who shared his extensive wisdom and offered indispensable advice on my research. I fondly remember how we mutually supported each other through the tumultuous periods of our research and personal lives.

I'm profoundly grateful to my thesis committee members and examiners, Professor Xiao-Wen Chang and Professor Marc G. Bellemare, Professor Benjamin Fung, and Professor Bang Liu, for their roles in my committee and their insightful critiques and assessments of my academic performance. I extend my gratitude to Professor Geoffrey Gordon for his

invaluable guidance in my research. Although he was unable to serve on my committee, his direction was instrumental in shaping my work.

I would also like to acknowledge my past and present labmates in the Cyber-Physical System Lab at McGill University. Our valuable discussions over the years have been instrumental in shaping my research journey. Their insightful contributions and the intellectually enriching environment we cultivated together are truly appreciated. In particular, I'd like to thank Dr. Qinglong Wang, Prof. Chen Ma, Dr. Jing Chen, Dr. Hang Li, Xuan Li, Dan Liu, Fuyuan Lv, Can Chen, Haolun Wu, Erqun Dong, Hanqing Zhao, Ye Yuan, Dun Yuan, Junliang Luo, Zonglun Wu, Shuhao Zheng, among others. They have created a vibrant research atmosphere and treasured memories for my Ph.D. journey.

Finally, my heartfelt thanks to my family: my wife, Xing Pan, my mother, Jingping Wang, and my father, Qiang Kang. Their unwavering love and support have been my anchor throughout this journey. Meeting my wife in the first year of my Ph.D. was a blessing. Her unwavering faith in me, even in the face of setbacks and failures, was a source of immense strength. Words fall short to express my gratitude and love for her. My parents, despite the distance and their own struggles, have been pillars of support. As an only child, I regret not being able to be there for them as much as I would have liked. Their love and sacrifice, particularly my mother's courage in carrying burdens alone to spare me worry, leave me profoundly grateful.

Regrettably, I cannot individually acknowledge all the remarkable individuals who have

provided invaluable help and support throughout my Ph.D. journey due to space constraints. Nevertheless, I am profoundly grateful for their presence in my life and the immeasurable contributions they have made. Their unwavering support and generosity with their time have been truly invaluable. It is my heartfelt wish that they enjoy lives filled with happiness, good health, and prosperity.

Contribution of Authors

In the following, I provide a contribution statement of co-authors regarding Chapters 4, 5, 6, and 7.

Chapter 4 is based on the work [1] that is published in the proceedings of IEEE Global Communications Conference (GLOBECOM 2022). I proposed the idea, implemented the algorithm, and wrote the paper draft. Ju Wang helped write the introduction and revise the paper. Chengming Hu helped write the related work. Xue Liu and Gregory Dudek helped revise the paper. All authors participated in the discussion of experimental results analysis.

Chapter 5 is based on the work [2] that is published in the proceedings of the 6th Conference on Robot Learning (CoRL 2022). I proposed the idea, implemented the algorithm, and wrote the paper draft. Miao Liu helped polish the idea and revise the paper. Jie Fu helped polish the idea, write the introduction and revise the paper. Abhinav Gupta, Christopher Pal, and Xue Liu helped revise the paper. All authors participated in the discussion of experimental results analysis.

Chapter 6 is based on the preprint work [3]. I proposed the idea, implemented the

algorithm, and wrote the paper draft. Romain Laroche, Xingdi Yuan, and Adam Trischler, Xue Liu, and Jie Fu helped revise the paper. All authors participated in the discussion of experimental results analysis.

Chapter 7 is based on the work [4] that is published in the proceedings of IEEE International Conference on Communications (ICC 2021). I proposed the idea, implemented the algorithm, and wrote the paper draft. Xi Chen helped write the introduction and revise the paper. Di Wu helped write the related work and revise the paper. Yi Tian Xu, Xue Liu, Gregory Dudek, Taeseop Lee, and Intaik Park helped revise the paper. All authors participated in the discussion of experimental results analysis.

Publications

In Conference Proceedings

1. **Kang, Jikun**, Di Wu, Ju Wang, Ekram Hossain, Xue Liu, and Gregory Dudek. "Multi-agent Attention Actor-Critic Algorithm for Load Balancing in Cellular Networks." In IEEE International Conference on Communications (ICC 2023).

2. **Kang, Jikun**, Miao Liu, Abhinav Gupta, Christopher Pal, Xue Liu, and Jie Fu. "Learning Multi-Objective Curricula for Robotic Policy Learning." In Conference on Robot Learning (CoRL 2022), pp. 847-858. PMLR, 2023.

3. **Kang, Jikun**, Ju Wang, Chengming Hu, Xue Liu, and Gregory Dudek. "A Generalized Load Balancing Policy With Multi-Teacher Reinforcement Learning." In GLOBECOM 2022-2022 IEEE Global Communications Conference, pp. 3096-3101. IEEE, 2022.

4. Wu, Di, **Kang Jikun**, Yi Tian Xu, Hang Li, Jimmy Li, Xi Chen, Dmitriy Rivkin et

al. "Load balancing for communication networks via data-efficient deep reinforcement learning." In 2021 IEEE Global Communications Conference (GLOBECOM), pp. 01-07. IEEE, 2021.

5. **Kang, Jikun**, Xi Chen, Di Wu, Yi Tian Xu, Xue Liu, Gregory Dudek, Taeseop Lee, and Intaik Park. "Hierarchical policy learning for hybrid communication load balancing." In ICC 2021-IEEE International Conference on Communications, pp. 1-6. IEEE, 2021.

6. Hu, Chengming, Xi Chen, Ju Wang, Hang Li,**Kang Jikun**, Yi Tian Xu, Xue Liu et al. "AFB: Improving Communication Load Forecasting Accuracy with Adaptive Feature Boosting." In 2021 IEEE Global Communications Conference (GLOBECOM), pp. 01-06. IEEE, 2021.

7. Gupta Abhinav, Chakravoty Jhelum, **Kang, Jikun**, and Precup Doina. Multi-agent option critic framework. Neurips RL Workshop, 2020.

Journal Articles

1. T. Chen, S. Bu, X. Liu, J. Kang, F. R. Yu and Z. Han, "Peer-to-Peer Energy Trading and Energy Conversion in Interconnected Multi-Energy Microgrids Using Multi-Agent Deep Reinforcement Learning," in IEEE Transactions on Smart Grid, vol. 13, no. 1, pp. 715-727, Jan. 2022, doi: 10.1109/TSG.2021.3124465.

In Submission

1. **Kang, Jikun**, Romain Laroche, Xindi Yuan, Adam Trischler, Xue Liu, and Jie Fu. "Think Before You Act: Decision Transformers with Internal Working Memory." arXiv preprint arXiv:2305.16338 (2023).

2. **Kang Jikun**, Ju Wang, Di Wu, Xue Liu and Gregory Dudek, Hyper-Decision Transformer: Network Load Balancing via Sequence Modeling, In Submission of GLOBECOM 2023.

Contents

List of Figures

List of Tables

List of Acronyms

ACL	automatic curriculum learning.
AULB	active user equipment load balancing.
CL	curriculum learning.
COMA	conterfactual multi-agent.
CTDE	centralized training decentralized execution.
D2D	Device-to-Device.
DDPG	deep deterministic policy gradient.
DQN	deep Q-Network.
DRL	deep reinforcement learning.
DRQN	deep recurrent Q-Network.
DT	Decision Transformer.
DT-Mem	Decision Transformer with memory.
HAC	hierarchical actor-critic.
HDT	hyper-Decision Transformer.

HPL	hierarchical Policy Learning.
HRL	hierarchical reinforcement learning.
IoT	Internet of Things.
IQL	independent Q-Learning.
IULB	idle user equipment load balancing.
KD	knowledge distillation.
K-NN	k-nearest neighbor.
LLM	large language model.
LoRA	low-rank adaptation.
LSTM	long-short term memory.
MADDPG	multi-agent DDPG.
MDP	Markov decision process.
MDT	multi-game Decision Transformer.
ML	machine learning.
MOBA	multi-teacher model-based reinforcement learning.
MOC	multi-objective curricula.
MPC	model predictive control.
MSE	mean squared error.
NN	neural network.
NTM	Neural Turing Machine.

PDT	prompt Decision Transformer.
PER	prioritized experience replay.
RL	reinforcement learning.
SARSA	state-action-reward-state-action.
SOTA	state-of-the-art.
UE	user equipment.

Chapter 1

Introduction

Building upon the tremendous achievements made in the field of deep neural networks, deep reinforcement learning (DRL) has established itself as a potent tool in a plethora of decision-making tasks. With its unique ability to learn from interacting with an environment and optimize a sequence of decisions, DRL has managed to impressively demonstrate its superior performance across a multitude of diverse domains.

Most notably, in the realm of video games, DRL has been used to achieve performance levels that match or even surpass human abilities. As documented in the groundbreaking work of [5], deep reinforcement learning algorithms have successfully mastered a variety of games, displaying an understanding and strategic aptitude that matches human game players.

Similarly, the strategic game of Go, once thought to be impervious to machine learning

due to its complexity, was eventually mastered by a DRL algorithm as demonstrated by [6]. Furthermore, DRL has also showcased its capability in managing complex continuous control tasks, a feat reported in the work of [7], where DRL agents proved adept at handling high-dimensional control problems.

Looking ahead, researchers are focus on the potential applications of DRL in a wide array of fields. The applicability of DRL extends to areas such as autonomous driving [8], where it can be used to optimize decision-making processes in complex, dynamic environments. Furthermore, it holds significant promise in networking [9], where it could efficiently manage network traffic and optimize resource allocation. Also, in the field of robotics [10], DRL can be utilized to empower robots to learn complex tasks through interaction with their environments.

However, as promising as DRL may seem, the application of these algorithms in real-world scenarios presents several notable challenges. The real world is fraught with stochasticity, with outcomes subject to random variations. It's also open-ended, with no predefined or distinct states, and constantly changes over time. These characteristics introduce a high level of complexity and uncertainty, which can significantly limit the effectiveness and potential application of DRL algorithms. Therefore, the journey to fully realize the promise of DRL in real-world applications requires more research to address these inherent challenges.

1.1 Challenges

Despite extensive research on deep reinforcement learning over the past decades, most studies have been restricted to games or simulator benchmarks, impeding its application in real-world scenarios. The main challenges can be summarized as follows: data efficiency, training complexity, model generalization, and safety concerns. These challenges are further illustrated below:

- *Data Efficiency.* DRL algorithms typically require a large amount of data to learn effectively. In real-world applications, obtaining sufficient data can be costly, time-consuming, or even impractical. Collecting real-world data may involve physical interactions, such as robots, where each interaction takes time and resources. In this thesis, we apply DRL in network load-balancing problem. Recently, reinforcement learning (RL) based methods [4], especially deep reinforcement learning (DRL), illustrate their ability in finding an optimal load balancing policy. However, existing RL-based methods normally learn from a specific traffic pattern, which requires a large amout of data to be trained on a wide variety of situations. Although one can learn a set of policies and pick a specific policy for each traffic pattern [11], data efficiency is the main challenge preventing it from applying in real-world.

- *Training Complexity.* DRL often requires a significant number of interactions with the environment to achieve good performance. This training complexity can be a challenge

in real-world scenarios where interactions may be expensive, risky, or time-sensitive. In this thesis, we adopt the automatic curriculum learning (ACL) to train agents' policies progressively. Oftentimes, only a single ACL paradigm (*e.g.*, generating subgoals) is considered. However, it remains an open question whether different paradigms are complementary to each other and if yes, how to combine them in a more effective manner similar to how the "rainbow" approach of [12] has greatly improved DRL performance in Atari games.

- *Model Generalization.* DRL algorithms often struggle with generalizing from the training environment to novel, unseen situations. Real-world applications often require agents to adapt and perform well in varying environments, which may differ significantly from the training setup. Ensuring generalization and robustness of learned policies is a significant challenge. Recently, with the tremendous success of large language model-based (LLM-based) foundation models [13]–[16], an increasing number of researchers have focused on LLM-based decision-making agents. As shown with GPT-3 [13] and follow-up work [17], [18], the generalization of these LLMs depends significantly on the model size, *i.e.* the number of parameters. This is partly because neural network parameters act as implicit memory [19], enabling models to "memorize" a huge amount of training data by fitting these parameters. However, relying purely on the scale has limits, practical and otherwise: there are economic and ecological costs, it reduces accessibility, and more efficient uses of scale might

improve performance further.

- *Safety Concerns.* In real-world applications, the actions taken by an RL agent may have physical or societal consequences. Ensuring safety behaviour becomes critical. For example, In the domain of autonomous driving, an RL agent's decision-making directly impacts passenger safety and traffic conditions. Incorrect actions can lead to accidents, posing significant safety risks. In this thesis, we focus on safety problems in network load-balancing problems. Existing work shows that either active user equipment load balancing (AULB) or idle user equipment load balancing (IULB) performs well individually. Thus, a natural question to ask is that *can we combine both AULB and IULB to better balance the communication load?* The answer seems to be a straightforward "yes" at first glance. However, it is actually difficult to support an affirmative answer, due to the challenge that the actions of AULB may conflict with the actions of IULB (and vice versa), resulting in unexpected degradation on system performance and safety issues.

1.2 Motivations

To effectively tackle the aforementioned challenges, it is imperative to employ more advanced techniques.Inspired by the latest advancements in machine learning, our approach involves carefully selecting appropriate methods and seamlessly integrating them with existing RL

algorithms Thus, we propose several different approaches to address the above-mentioned challenges:

- *Knowledge Distillation Enhanced Data Efficiency.* We adopt knowledge distillation (KD) to address the data efficiency challenge. The advantages can be summarized in two folds: 1) KD can enable the student model to learn more efficiently from the teacher model's experience, requiring fewer training samples from the environment and improving the data efficiency. 2) The student model can benefit from the teacher model's experience, particularly useful in cases where the teacher model has undergone extensive training or has been trained on a wide range of tasks or environments.

- *Curriculum learning Enhanced Training Complexity.* To improve agent training in complex real-world environments, we adopt a curriculum learning idea to generate multiple curricula progressively. This is because: 1) In curriculum learning, agents start learning from simple tasks. This process can make it easier to find initial policies that yield positive rewards, helping to bootstrap the learning process. 2) By progressively learning from simpler to more complex tasks, the model can converge faster, as it doesn't have to learn everything at once. This technique can reduce training complexity and time.

- *Sequence Modeling Enhanced Generalization.* Our motivation comes from the recent success of large language model-based (LLM-based) foundation models [13]–[16].

Recently, [20] and [21] treat the RL problem as a sequence modeling problem and proposed a Transformer-based architecture to solve it with offline RL. These findings inspired researchers to develop more advanced Transformer-based RL methods. Thus, we want to leverage generalization of LLM and propose an transformer architecture to solve various tasks.

- *Hierarchical Structure Enhanced Safety Actions.* We adopt a hierarchical structure to overcome the safety concerns. The main reason is that, with hierarchical DRL, high-level policies make strategic decisions while low-level policies execute actions. This separation can ensure safety-critical decisions are made at an appropriate level of abstract instructions.

1.3 Contributions

Inspired by the motivations listed in Section 1.2, we investigate and design several algorithms, which address challenges listed in Section 1.1 accordingly. In particular, the major contributions of this thesis are listed as follows:

- In Chapter 4, we introduce a multi-teacher knowledge distillation approach to learn the state transition function and reward function in model-based RL. The key is that different teachers represent different traffic patterns, and can learn various system models. By distilling and transferring the teacher knowledge, the student network

is able to learn a generalized system model that covers different traffic patterns and unseen situations, which decreases the number of data samples required for training. Moreover, to improve the robustness of multi-teacher knowledge transfer, we learn a set of student models and use an ensemble method to jointly predict system dynamics.

- In Chapter 5, we propose a unified automatic curriculum learning framework to create multi-objective but coherent curricula that are generated by a set of parametric curriculum modules. Each curriculum module is instantiated as a neural network and is responsible for generating a particular curriculum. In order to coordinate those potentially conflicting modules in a unified parameter space, we propose a multi-task hyper-net learning framework that uses a single hyper-net to parameterize all those curriculum modules. We evaluate our method on a series of robotic manipulation tasks and demonstrate its superiority over other state-of-the-art automatic curriculum learning (ACL) methods in terms of training efficiency and final performance.

- In Chapter 6, we propose an internal working memory module to store, blend, and retrieve information for different downstream tasks. Specifically, we instantiate the internal working memory as a matrix and its functioning entails two primary steps: **memory update** and **memory retrieval**. The memory update involves modifying or replacing existing information. This enables the system to keep track of changes, maintain task-relevant information, and facilitate decision-making. Memory retrieval

refers to the process of accessing and recovering stored information. It involves bringing relevant information back to condition decision-making. We use content-based addressing [22]–[24] to locate the memory position to update or retrieve from. To update the memory, we first map the input sequence and memory into three entities: query, key, and value. Next, we use an attention-based mechanism to calculate the correlations between the input and memory, and then we use the attended weight of the input sequence to update the memory. To retrieve, we read from the updated memory at the content-based address. Evaluation results show that the proposed method improves training efficiency and model generalization in both Atari games and meta-world object manipulation tasks. Moreover, we demonstrate that memory fine-tuning further enhances the adaptability of the proposed architecture.

- In Chapter 7, we propose a Hierarchical Policy Learning (HPL) framework, which coordinates the actions between different policies with a two-level learning structure. Concretely, the upper level adjusts one level of actions, and the lower level controls the other level of actions. The upper level aims to optimize the system performance directly as an RL reward, and at the same time, learns to set a subgoal for the lower level. This subgoal is a desired RL state, which further improves the upper-level reward (and yet cannot be achieved with only upper-level actions). By approaching this subgoal, the lower level 1) indirectly enhances the system performance, and 2) is enforced to align

with the upper level. In this way, collaboration is established between two levels of action, eliminating potential conflicts and addressing safety concerns.

Chapter 2

Background

In this chapter, we introduce the background of reinforcement learning (RL), the problem settings in the RL research community, and the preliminaries of mechanisms adopt in this thesis.

2.1 Overview

Reinforcement learning can be categorized in several ways in terms of the nature of the learning process, the feedback received, and the environment. RL can be briefly divided into the following categories:

- *Model-based vs. Model-free*: In model-based RL, the agent creates a model of the environment to make decisions, which involves predicting the next state and the reward.

In contrast, model-free RL directly learns a policy or a value function without explicitly modelling the environment.

- *Value-based vs. Policy-based* In value-based RL, the agent aims to learn the value of each state or state-action pair, which represents the expected cumulative future reward. The policy (i.e., the strategy for choosing actions) is then derived from these values. In policy-based RL, the agent directly learns the policy without explicitly learning the value function.

- *On-policy vs. Off-policy* In on-policy learning, the agent learns the value of the policy currently being used to make decisions. In off-policy learning, the agent learns the value of one policy while following another policy. Off-policy learning allows an agent to learn from previous experiences, even as its policy evolves.

- *Single-agent vs. Multi-agent* In single-agent RL, the agent learns to optimize its decisions in an environment where it is the only entity making decisions. In multi-agent RL, multiple agents interact with the environment and each other, leading to more complex dynamics.

These categories are not mutually exclusive and many RL algorithms fall into different categories. This thesis mainly focuses on single-agent settings and leaves the multi-agent settings in the future research.

2.2 Problem Setting

Formally the single-agent RL problem is formulated as a Markov decision process (MDP) M. A MDP can be described by a tuple $M = (S, A, P, R, \gamma)$:

- S: is the state set. The state of time t is denoted as $S_t = s$.
- A: is the action set. The action of time t is $A_t = a$.
- P: the state transition probability function, where $P(S_{t+1} = s'|S_t = s, A_t = a)$ maps a state–action pair at time t to a probability distribution over states at time $t + 1$, such that $P : s \times a \times s' \to [0, 1]$.
- R: the reward function, where $R \times A \to \mathbb{R}$
- γ: is the discount factor, where $\gamma \in [0, 1)$.

The goal reinforcement learning is to learn a set of agent policies $\{\pi^a\}_{a=1,\dots,A}$ that maximise the total expected return per episode $J = \mathbb{E}_{\tau\sim P(\tau|\pi^a)}[\sum_t \gamma^t r_t]$. In deep reinforcement learning (DRL), optimization involves training neural networks that represent policies and value functions.

A policy $\pi : A = \pi(S)$ specifies a way of behaving, and its value function is the expected return obtained by following policy π. The value function V_π obeys the following Bellman equations:

$$V_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_\pi(s')),$$

where s' is the state following state s. The policy gradient theorem provides the gradient of the expected discounted return from an initial state $d(s_0)$ with respect to a parameterized stochastic policy π_θ:

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_s d(s;\theta) \sum_a \frac{\partial \pi(a|s)}{\partial \theta} Q_\pi(s,a),$$

where we simply write π for π_θ for ease of notation and $d(s:\theta) = \sum_{s_0} d(s_0) \sum_{t=0}^{\infty} \gamma^t P^\pi(S_t = s|S_0 = s_0)$ is the discounted state occupancy measure.

2.3 Preliminaries

2.3.1 Knowledge Distillation

To improve the performance of the student network, [25] introduces a knowledge distillation framework, which uses the supervision knowledge distilled from teacher networks. In general, the student network is trained to have similar output distribution with regard to teacher networks. The output of the student network is regulated to be close to the ground truth labels as well as the outputs of teacher networks.

$$\mathcal{L}_{KD}(\theta_S) = \mathcal{H}(y, f_{\theta_S}) + \alpha \mathcal{H}(f_{\theta_T}, f_{\theta_S}),$$

where $\mathcal{H}(\cdot,\cdot)$ is the cross-entropy, α is a hyper-parameter that regularizes the second term, and y stands for the ground-truth label. In addition, f_{θ_S} and f_{θ_T} are student network and teacher network respectively.

2.3.2 Curriculum Learning

Automatic curriculum learning (ACL) is a learning paradigm where an agent is trained iteratively following a curriculum to ease learning and exploration in a multi-task problem. Since it is not feasible to manually design a curriculum for each task, recent work has proposed to create an implicit curriculum directly from the task objective. Concretely, it aims to maximize a metric P computed over a set of target tasks $T \sim \mathcal{T}_{target}$ after some episodes t'. Following the notation in [26], the objective is set to: $\max_{\mathcal{D}} \int_{T\sim\mathcal{T}_{target}} P_T^{t'} dT$, where $\mathcal{D} : \mathcal{H} \to \mathcal{T}_{target}$ is a task selection function. The input $\mathcal{H}$ can consist of any information about past interactions. For example, in our experiments, the history consists of the last state of the episode, and the output of $\mathcal{D}$ is a sequence of generated episodes, which contain different desired goals, initial states, rewards, etc.

2.3.3 RL as a Sequence Modeling Problem

We formulate the RL problem as a Markov decision process (MDP) problem with tuples $\mathcal{T} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$: where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ the set of actions, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to (0, 1)$ the transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\gamma \in [0, 1)$ the discount

factor. In addition, $\pi(\cdot\,;\phi_\pi)$ designates a policy parameterized by ϕ_π, and $\pi(a|s;\phi_\pi)$ denotes the probability of choosing action $a \in \mathcal{A}$ given a state $s \in \mathcal{S}$. Here, we consider a transfer learning problem, where a pre-trained model is used as a starting point for a new task that is related or similar to the original task on which the model was trained. The idea behind transfer learning is to leverage the knowledge learned by the pre-trained model to improve performance on the new task, for which data may be lacking or inaccessible.

Formally, in the context of model evaluation, we can define a set of training tasks and testing tasks as T^{train} and T^{test}, respectively. These two sets deliberately have no overlapping tasks, but they may share the same or similar observation and action spaces. To be more specific, for each training task $\mathcal{T}^i \in T^{train}$, we have access to a large training dataset, which contains trajectories $\tau^{0:H} = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$, where H is the episode length. However, we assume access to only a small amount of data for the testing tasks.

Our goal is to evaluate the proposed model in two dimensions. First, we want to assess the model's **generalization**, which refers to its ability to solve the testing tasks within a finite time with no additional fine-tuning. Second, we want to test the model's **adaptability**, which refers to its ability to improve its performance on the testing tasks through fine-tuning on limited data after pre-training on separate tasks.

2.3.4 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) is a subfield of reinforcement learning that seeks to address complex problems by dividing them into a series of simpler sub-tasks. This approach is inspired by the hierarchical nature of human and animal decision-making processes, where complex tasks are naturally broken down into more manageable parts.

In traditional reinforcement learning, an agent aims to learn a single policy that maps every state to an optimal action. However, in complex environments with a large state-action space, this approach can be computationally expensive and slow to converge.

HRL solves this problem by constructing a hierarchy of policies at different levels of abstraction. At the highest level, a "meta-policy" makes broad, strategic decisions about what type of action to take. At lower levels, sub-policies handle the specifics of how to execute these high-level actions.

One way to do this is using options or skills frameworks. An option (or skill) is a pre-defined sequence of actions that achieve a particular sub-goal. Instead of choosing from individual actions at each step, the high-level policy chooses from these options, significantly simplifying the decision-making process.

However, HRL also has challenges such as how to define the hierarchy, how to coordinate the learning between different levels, and how to handle the trade-off between exploration and exploitation at different levels. Despite these challenges, HRL remains a promising approach for handling complex reinforcement learning problems.

Chapter 3

Related Work

In this chapter, we provide a comprehensive review of the related work in the field of RL algorithms, highlighting the various mechanisms that are adopted in this thesis. Additionally, we focus on the specific RL scenario of network load balancing problem, exploring the application of RL techniques in this domain. By examining the existing literature and drawing insights from previous research, we lay the foundation for our proposed methodologies and contribute to the advancement of RL for network load balancing.

3.1 Overview of Reinforcement Learning

Contrary to supervised or unsupervised learning, reinforcement learning is primarily concerned with deciphering how agents should operate within an environment to maximize

their accumulated reward [27]. It has been successfully leveraged across various domains, ranging from Atari games [28] and robotics [29], to natural language processing [30] and healthcare [31]. In this thesis, we mainly focus on the following categories of RL methods.

3.1.1 Model-free Reinforcement Learning

Model-free RL is an extensive and active research area that has made numerous advancements over the past few decades. The primary advantage of model-free RL is that it doesn't require learning the system dynamics (i.e., state transition function and reward function). Instead, the agent treats the environment as a black box and approximates the value or policy function by interacting with it. Broadly speaking, there are two primary categories of model-free RL algorithms: value-based and policy-based methods.

Value-based methods stand for a series of RL algorithm that learns the state-value function or action-value function. The policies are inferred by choosing actions that collect the largest return value at the current state. The classical value-based method is Q-learning, which learns optimal action-value functions directly from trajectories gained by interacting with the environment [32]. State-action-reward-state-action (SARSA) is designed to learn the Q-value associated with taking a specific action in a given state and following the current policy thereafter [33]. Deep Q-Network (DQN) utilizes deep neural networks (DNNs) to approximate the Q-function. DQN also introduces replay buffer and target network techniques to overcome the challenges of applying DNNs in RL [28]. Double

DQN is proposed to reduce the overestimation bias of DQN by using two separate networks for action selection and policy evaluation, respectively [34]. The following work Prioritized Experience Replay (PER) aims to improve learning efficiency by prioritizing the stored experience in the buffer, instead of uniformly sampling from the replay buffer [35]. The motivation behind Dueling DQN is to separately represent the state-value and action-advantage functions in the network architecture, to better understand the underlying state values without the need of learning all action values [36]. To solve the partial observable environment, Deep Recurrent Q-Network (DRQN) incorporate recurrent neural networks (e.g., LSTM) into the DQN architecture to allow the agent to maintain a memory of past observations [37]. Compared to the epsilon-greedy approach used in DQN, Noisy DQN aimed to encourage the agents' exploration by adding parametric noise to the weights of the network [38]. Rainbow DQN is presented to combine the above-mentioned improvements (including DDQN, PER, Noisy DQN and Dueling Networks) into a single architecture [12].

3.1.2 Model-based Reinforcement Learning

Model-based reinforcement learning algorithms are famous for solving real-world sequential decision-making problems due to their data efficiency [39]. Usually, one cannot directly obtain the environment model (i.e., state-transition function and reward function), thus there are plenty of ways to approximate the dynamic model by treating the system model

as a black box. Among all the approximators, Neural Networks (NNs) based approximators are widely used in model-based reinforcement learning, due to its asymptotic performance of high-capacity approximate function [40]. Moreover, NNs can scale to high dimensional control problems with better sample efficiency [41], [42]. The classical model-based algorithm is Dyna-Q, which integrates planning, acting, and model learning into a unified system [43]. The motivation is to improve the agent's policy by simulating the next states and rewards from the learned model, thus decreasing the interactions with the environment. Lately, model predictive control (MPC) is introduced for planning by optimizing actions over a finite horizon and executing only the first action in the sequence. The motivation is to create a feedback system that plans and re-plans at each time step to respond to environmental changes [44]. The World Models framework learns a model of the environment and utilizes the model to train the policy [45].

3.1.3 Hierarchical Reinforcement Learning

Hierarchical RL algorithms originate from the ideas of options framework [46]. The Options Framework introduces temporal abstractions into reinforcement learning through options, which are sub-policies covering multiple time steps. The motivation is to improve learning and planning by structuring the decision process on multiple time scales. Lately, this options framework is extended to the Option-Critic framework, which integrates the learning of options and the policy over options into a unified learning process [47]. Hierarchical-DQN (h-

DQN) adopt the hierarchical options framework idea to the DQN algorithm [48]. In h-DQN, one DQN plays the role of selecting goals and another DQN take actions to achieve these goals. This method brings the benefits of hierarchical decision-making to DRL. Inspired by h-DQN, hierarchical actor-critic (HAC) is proposed to improve learning efficiency for sparse and delayed rewards tasks [49]. HAC allows for flexible goal specifications and can learn high-level strategies to achieve complex tasks.

3.1.4 Reinforcement Learning via Sequence Modeling

Transformer [50] is a powerful architecture designed for sequence modeling. Owing to the capabilities that emerge as model and data size scale up, the Transformer has become a foundational model in several domains, including natural language processing [13], [14], [16] and computer vision [15]. However, applying Transformers in reinforcement learning settings, such that they generalize to multiple tasks, remains an open problem.

Recently, [20] and [21] treat the RL problem as a sequence modeling problem and proposed a Transformer-based architecture to solve it with offline RL. These findings inspired researchers to develop more advanced Transformer-based RL methods. Subsequent efforts mainly focus on two aspects: generalization and adaptability. To improve model online adaptability, [51] propose the Online Decision Transformer (Online DT), which utilizes the maximum-entropy idea to encourage pre-trained policies to explore during a phase of online adaptation. To improve offline adaptation, [52] propose a

Hyper-network-based module that helps DT adapt to unseen tasks efficiently. To facilitate task adaptation, [53] introduce the prompt-based DT, which selects short trajectories to use in a task prompt in analogy with in-context learning for large language models. Furthermore, [54] propose a multi-game DT (MDT), which use the expert action inference to consistently produce actions of highly-rewarding behavior. MDT demonstrating that DT can generalize to various Atari games with human-level performance. We argue that the generalization of the above-mentioned works relies on the size of models and does not learn the data efficiently. To address this issue, we introduce a working memory module that can store, blend, and retrieve training information for better model and training efficiency.

3.2 Related Mechanisms

3.2.1 Knowledge Distillation

Knowledge distillation is first proposed by [25] and aims to help the training process of a smaller student network under the supervision of a larger teacher network. FitNets [55] encourage an intermediate layer of the student network to have the ability to match the outputs of some intermediate layers of the teacher network. The relationships among different neural layers and neurons [56], [57] are also considered as knowledge distilled by teacher models. To better transfer knowledge, multiple teacher networks have been introduced in the knowledge distillation framework where a student network can simultaneously receive

knowledge distilled from multiple teacher networks.

The average ensemble of logits is a commonly-used approach in multi-teacher knowledge distillation. In such a setting, a student network is encouraged to learn the average softened output of multiple teacher networks' logits via minimizing the cross-entropy loss, and the average softened output serves as the incorporation of multiple teacher networks in the output layer [58], [59]. In [60], authors formulate the teacher selection problem under an RL framework, where each teacher network is assigned an appropriate weight based on various training samples and the outputs of teacher networks.

3.2.2 Transformer-based Reinforcement Learning methods

Transformer [50] is a powerful architecture designed for sequence modeling. Owing to the capabilities that emerge as model and data size scale up, the Transformer has become a foundational model in several domains, including natural language processing [13], [14], [16] and computer vision [15]. However, applying Transformers in reinforcement learning settings, such that they generalize to multiple tasks, remains an open problem.

Recently, [20] and [21] treat the RL problem as a sequence modeling problem and proposed a Transformer-based architecture to solve it with offline RL. These findings inspired researchers to develop more advanced Transformer-based RL methods. Subsequent efforts mainly focus on two aspects: generalization and adaptability. To improve model online adaptability, [51] propose the Online Decision Transformer (Online DT), which

utilizes the maximum-entropy idea to encourage pre-trained policies to explore during a phase of online adaptation. To improve offline adaptation, [52] propose a Hyper-network-based module that helps DT adapt to unseen tasks efficiently. To facilitate task adaptation, [53] introduce the prompt-based DT, which selects short trajectories to use in a task prompt in analogy with in-context learning for large language models. Furthermore, [54] propose a multi-game DT (MDT), which use the expert action inference to consistently produce actions of highly-rewarding behavior. MDT demonstrating that DT can generalize to various Atari games with human-level performance. We argue that the generalization of the above-mentioned works relies on the size of models and does not learn the data efficiently. To address this issue, we introduce a working memory module that can store, blend, and retrieve training information for better model and training efficiency.

3.2.3 Working memory

In the context of machine learning, there is a long history of neural network-based models that incorporate memory mechanisms [61]–[69]. Generally, this research aims to enhance the capacity of neural networks to store and manipulate information over extended periods of time, leading to improved performance on a range of tasks. It often takes inspiration from human cognitive function. Most salient to our work, [22] merge concepts from Turing machines and deep learning in “Neural Turing Machines” (NTMs), neural networks that include a content-addressable matrix memory space for storing and updating information

throughout time. They show NTMs to be effective for various algorithmic tasks. Contemporaneously, [70] introduce "memory networks," which use a content-addressable matrix memory store and retrieve information from previous computational steps to facilitate complex reasoning and inference tasks. [71] propose a rapidly adaptable neural memory system, which they instantiate as a feedforward neural network trained by metalearning. They evaluate the memory's effectiveness in a simple RL setting, maze exploration, and on various NLP tasks. This work can be seen as a precursor to our use of LoRA to adapt the working memory module. More recently, [72] utilize the "global workspace" theory from cognitive science, which posits that different input entities share information through a common communication channel. The proposed shared global workspace method utilizes the attention mechanism to encourage the most useful information to be shared among neural modules. It is closely related to working memory and inspires us to explore how an explicit working memory can improve the generalization of Transformer-based models. An upshot of our work is that it may be valuable to revisit earlier memory-augmentation methods in light of more powerful foundation models.

3.2.4 Curriculum Learning

Curriculum learning (CL) is an important learning strategy that can be applied to RL. The reason is that CL helps decompose a hard task into several subtasks from simple to complex. The intuition behind CL is inspired by the human learning process, in which one often starts

with simpler tasks before gradually taking on more difficult ones [73].

As pointed out in [74], in the context of RL, CL can control five types of elements in RL tasks: which are goal generation, reward shaping, environment generation, initial state generation, and opponents generation. In [75], authors show the benefits of a diverse set of starting states to guide the learning processing in manipulation tasks. Similarly, [76] introduces a reverse curriculum generation strategy for robot locomotion tasks.

However, one of the key challenges in applying CL is how to define an effective and scalable curriculum. To address this issue, recent works focus on automatic curriculum learning (ACL), which adjusts curriculum based on the agents' performance [77]. Specifically, the teacher-student framework is proposed for ACL, where the teacher plays the role as curriculum generator and the agent is the student [74].

While automatic curriculum learning has demonstrated successes, all of the above work only considers single curriculum generation and doesn't show the potential of multiple curricula generation. In this thesis, we aim to generate multiple curricula in a unified framework, where all curricula contribute to the agents' policy learning.

3.3 Applied Reinforcement Learning

3.3.1 RL for Network Load-balancing

Load Balancing (LB) has been important topic for cellular network performance and efficiency. Both Active User Equipment LB (AULB) and Idle User Equipment LB (IULB) have been explored in the literature. To control the LB actions, existing methods employ either rule-based algorithms or RL-based algorithms. For rule-based algorithms, a set of different LB strategies or parameters are pre-defined. Control actions are taken based on the serving cell and the neighboring cells' signal measurements to minimize the call dropping rate, usually with a fixed size of control steps [78]. The adaptive step size has also been studied for mobile load balancing in [79]. In general, the performance of rule-based methods is limited by the mismatch between pre-define rules and the ever-changing environment.

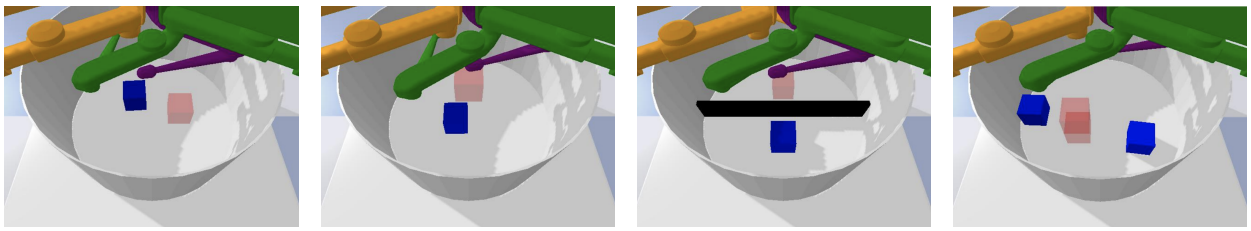
RL aims to learn a control policy to maximize some long-term expected reward, by interacting with the environment. It also been used to balance the communication load. In [80], Q learning is applied to deal with mobility load balancing. Compared with rule based method, Q learning shows better performance on improving system performance in changing environments. It is also shown that the number of unsatisfied users could be improved by Q learning based load balancing method [81]. Q learning based method has also been used to improve the load distribution and the system robustness in [82]. Fuzzy Q

learning has been proposed in [83] to better deal with uncertainty in the observations.

However, Q learning methods, as well as classic joint control methods (e.g., OnCAR [84]), suffer from the curse of dimensionality. To address this issue, Deep RL (DRL) methods use deep neural networks as function approximators, and illustrate impressive performance for different control tasks. Specifically, it has been applied in [85] to learn multiple LB policies for a more robust cellular network. In [86], DRL also illustrate its advantage in balancing the load in Device-to-Device (D2D) networks. In addition, DRL has been employed for the LB in Internet of Things (IoT) networks [87].

3.3.2 Robotics Control

The second application we are interested in is robotics control. In this proposal, we adopt a novel benchmark CausalWorld [88], as this environment enables us to easily design and test different types of tasks in a fine-grained manner. It is worth noting that this environment also provides a wrapper that makes the environment execute actions on the real robot, which can be used in sim2real experiments. We choose five out of the nine tasks introduced in CausalWorld since the other four tasks have limited support for configuring the initial and goal states, which are shown in Figure 3.1. Specifically, we enumerate these five tasks here: (1) `Reaching` requires moving a robotic arm to a goal position and reaching a goal block; (2) `Pushing` requires pushing one block towards a goal position with a specific orientation (restricted to goals on the floor level); (3) `Picking` requires picking one block at a goal height



(a) Reaching/Pushing **(b)** Picking **(c)** Pick and Place **(d)** Stacking

Figure 3.1: Example tasks from the task generators provided in the benchmark. The goal shape is visualized in opaque red and the blocks in blue.

above the center of the arena (restricted to goals above the floor level); (4) `Pick And Place` is an arena divided by a fixed length block and the goal is to pick one block from one side of the arena to a goal position with a variable orientation on the other side of the fixed block; (5) `Stacking` requires stacking two blocks above each other in a specific goal position and orientation.

3.3.3 Arcade Learning Environment

The third application we focus on is Atari games. The Arcade Learning Environment (ALE), originally proposed by Bellemare [89], serves a dual purpose: it is both a challenging problem and a platform for evaluating general competency in artificial intelligence (AI). Atari 2600 games are ideal for AI agent assessment for three primary reasons: 1) the variety of games provides multiple distinct tasks, necessitating a general competence, 2) they offer interesting and human-engaging challenges, and 3) they are free from experimenter bias, being developed independently. The significance of ALE is evident

in the considerable attention it has garnered within the scientific community. There has been a surge in research papers utilizing ALE as a testing ground in recent years. Notable achievements include the development of Deep Q-Networks (DQN) [5], which was the first algorithm to reach human-level control in many Atari 2600 games, garnering widespread acclaim.

In this thesis, the offline datasets are derived from the work of [90]. The datasets, known as the DQN replay datasets, were generated by training multiple DQN agents, following the approach outlined by [5], across 60 Atari 2600 games. Each game involved training sessions of 200 million frames, adhering to a standard protocol with a frame skip of 4 and the inclusion of sticky actions. For each game, five unique agents were trained, each initialized randomly. Throughout the training process, comprehensive data comprising tuples of observation, action, reward, and subsequent observation were collected. This resulted in the creation of five distinct replay datasets for each game, cumulating in a total of 300 datasets. Remarkably, each individual game replay dataset is approximately 3.5 times the size of the ImageNet dataset. These datasets are particularly valuable as they encompass samples from a wide range of intermediate and diverse policies observed during the optimization process of the online DQN agents.

Chapter 4

Knowledge Distillation Enhanced Sample Efficiency

4.1 Introduction

Load balancing (LB) plays an important role in the cellular network, since it can maximize resource usage, minimize response time, and reduce computation overload [1]. To provide a balanced load, it is important to take a wide variety of network traffic patterns into consideration. Because users or user equipments (UEs) may consume different bandwidths due to different user habits and scenarios. For example, on workdays, users tend to use low-frequency bandwidth that can provide better connectivity for lightweight tasks such as mailing. On weekends, users are likely to consume high-frequency bandwidth, which can

provide higher throughput, to watch streaming shows for entertainment purposes. Hence, it is a challenge to come up with a generalized load balancing policy that can adapt to different traffic patterns.

Recently, reinforcement learning (RL) based methods [2], especially deep reinforcement learning (DRL), illustrate their ability in finding an optimal load balancing policy. However, existing RL-based methods normally learn from a specific traffic pattern, which cannot be applied to a wide variety of situations. Although one can learn a set of policies and pick a specific policy for each traffic pattern [1], it remains a challenge to deal with unseen new scenarios. There are two reasons. First, most existing RL-based load balancing algorithms are model-free methods, which learn directly from interactions of environments and treat system dynamics as a black box. When the traffic pattern changes, the trained agent cannot adapt to unseen patterns. Second, the model-free RL-based method requires huge amounts of data. As a result, the agent needs to make frequent interactions with the real environment, making it impossible or hard to train a generalized load balancing policy.

To develop a generalized load balancing policy, we propose a **M**ulti-teacher M**O**del **BA**sed Reinforcement Learning algorithm (MOBA), which embraces both the multi-teacher Knowledge Distillation[1] (KD) [3] and the model-based RL [4]. In particular, we first utilize the model-based RL to learn a set of traffic pattern models (including a state transition model and a reward model), which we called teacher models. Then, we

[1]Multi-teacher KD is first introduced for model compression. Recently, this technique has been widely used in different problems such as data privacy, natural language processing (NLP), and few-shot learning.

leverage the multi-teacher KD to transfer the knowledge of teacher models to a student model. Our key observation is that the ability of policy generalization is closely related to the learned student model because the learned student model will be generalized enough with enough transferred knowledge. Motivated by this, we learn a generalized load balancing policy by using multiple teacher models, where different teachers represent different traffic patterns. In contrast to existing methods, we show that distilling multi-teacher model knowledge not only improves the ability of model generalization but also avoids frequent interactions between the agent and environment, thus reducing the time cost and improving the data efficiency. Furthermore, we use an ensemble of student networks to predict system dynamics and improve the stability of the learned student model and policy. More importantly, our proposed framework is algorithm agnostic, which can be easily combined with other state-of-the-art policy learning methods.

We conduct experiments on a communication network simulator over four performance metrics. Compared to the state-of-the-art (SOTA) methods, MOBA improves system minimum throughput and total throughput by up to 28.6% and 23.2%, respectively. Results also show that MOBA can improve training efficiency by up to 64%.

To sum up, the contributions of this chapter are as follows:

- We introduce a multi-teacher knowledge distillation approach for a generalized load balancing policy. The learned student model not only has high accuracy but also is robust to traffic pattern changes.

- We utilize a model-based RL approach for load balancing, which improves training efficiency.

In this chapter, we delve into the experimental validation of the proposed models and algorithms. While the initial focus of our study centers on the load balancing problem in communication networks, it is crucial to illustrate the versatility and applicability of our models in varied contexts. To this end, we extend our experimentation to include tasks in robotics. This approach not only demonstrates the robustness of our models in handling complex load balancing scenarios but also their adaptability to different domains.

- **Load Balancing in Communication Networks:** The primary focus of our experiments lies in addressing the challenges in load balancing. We implement our models in simulated network environments to evaluate their efficacy in optimizing traffic distribution and managing network resources efficiently.

- **Robotics Tasks:** To further validate the generalizability of our models, we include experiments in robotics. These tasks involve complex control and decision-making scenarios, offering insights into how our models perform in dynamic and physically constrained environments.

This chapter thus presents a comprehensive evaluation of our models, showcasing their application not only in the specific context of load balancing but also in broader areas like robotics. Such a diverse experimental setup allows us to thoroughly assess the capabilities

and limitations of our proposed solutions.

4.2 Preliminaries

4.2.1 Model-based Reinforcement Learning

A discrete-time finite Markov decision process (MDP) is defined by the tuple $< S, A, p, r, \gamma >$. Here, S is the state space, A is the action space, $p : S \times A \times S$ is a state transition function, $r : S \times A \to \mathbb{R}$ is a reward function, and γ is the discount factor. The goal of reinforcement learning is to learn an agent policy π that can collect the largest expected return $\mathbb{E}[\sum_t^T \gamma^t r_t]$.

In contrast to model-free RL algorithms that don't model system dynamics (i.e. state transition function), model-based RL methods explicitly learn the state transition function p. The system dynamics can be treated as a black box and learned using various means. One of the methods we adopt in this chapter is a neural network function approximator. We use a function $f_\phi(s_{t+1}|s_t, a_t)$ that parameterized by ϕ to represent this approximator. We train this function by maximizing the log-likelihood of the state transition distribution.

4.2.2 Knowledge Distillation

To improve the performance of the student network, [3] introduces a knowledge distillation framework, which uses the supervision knowledge distilled from teacher networks. In general, the student network is trained to have similar output distribution with regard to teacher

networks. The output of the student network is regulated to be close to the ground truth labels as well as outputs of teacher networks.

$$\mathcal{L}_{KD}(\theta_S) = \mathcal{H}(y, f_{\theta_S}) + \alpha\mathcal{H}(f_{\theta_T}, f_{\theta_S}),$$

where $\mathcal{H}(y, f_\theta) = -\sum_{i=1}^{N} [y_i \log(f_{\theta_i}) + (1 - y_i) \log(1 - f_{\theta_i})]$ is the cross-entropy, α is a hyper-parameter that regularizes the second term, and y stands for the ground-truth label. In addition, f_{θ_S} and f_{θ_T} are student network and teacher network respectively.

4.2.3 Active UE Load Balancing (AULB) Feature

In this chapter, we consider a hybrid load balancing problem, which is consist of AULB and IULB. The load balancing solution AULB is based on active UE HandOver (HO). We propose a common Reference Signal Received Power (RSRP) based handover mechanism in LTE/5G networks for the purpose of generality, which can encompass several versions such as Cell Individual Offset (CIO) based handover or A2/A5 event based HO. Specifically, every UE compares the RSRP value of its serving cell to the values of its neighbors. If the following condition holds, then the active UE will be handed over to a neighboring cell, i.e., $RSRP_j > RSRP_i + \alpha_{i,j} + H$, where $RSRP_i$ represents the UE's RSRP from the serving cell i, $RSRP_j$ denotes the UE's RSRP from a neighboring cell j, $\alpha_{i,j}$ is the HO threshold from cell i to cell j, and H is the HO hysteresis. This HO threshold $\alpha_{i,j}$ is a pair-wise directional

variable (e.g., $\alpha_{i,j} \neq \alpha_{j,i}$).

4.2.4 Idle UE Load Balancing (IULB) Feature

Another load balancing approach IULB relies on idle UEs' Cell Re-selection (CR). When a UE is initially turned on, it goes into idle mode and then "camps" on a cell. A UE that is idle is ready to start a dedicated service or receive a broadcast service. The UE will generally stay in the same cell where it camped during the idle phase until it becomes active. An idle UE might use the CR technique to camp on another cell in order to stay connected while travelling. This CR procedure will be triggered, if the following condition holds for an idle UE: $RSRP_i < \beta_{i,j}$, and $RSRP_j > \gamma_{i,j}$, where $\beta_{i,j}$ and $\gamma_{i,j}$ are pairwise and directional RSRP thresholds to trigger CR from a camping cell i to a neighboring cell j.

4.3 Methodology

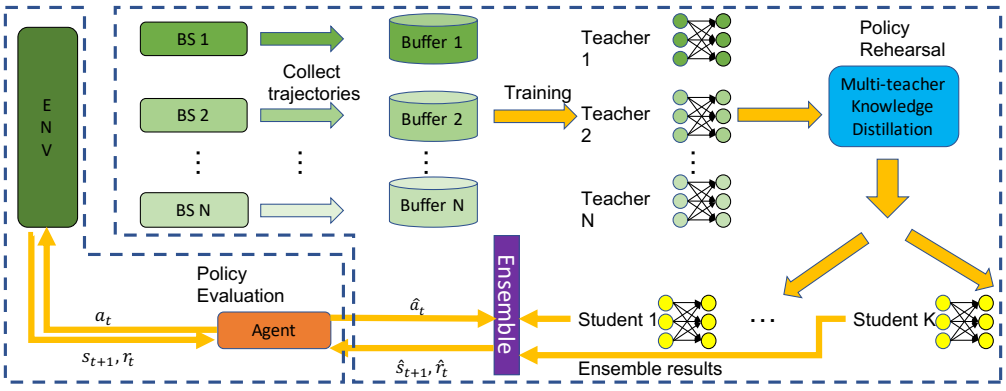


Figure 4.1: The architecture of our multi-teacher reinforcement learning method.

The overview of MOBA's architecture is depicted in Figure 4.1. At a high level, it consists of two parts, which are policy rehearsal and policy evaluation. Next, we detail our method.

4.3.1 Problem Statement

We intend to tackle a hybrid load balancing issue in this work, in which both AULB and IULB are used to achieve a balanced load and improved system performance. Formally, we define this hybrid load balancing problem as follows.

$$\max_{\{\alpha_{i,j}\},\{\beta_{i,j}\},\{\gamma_{i,j}\}} \mathbb{E}[\sum_{t=1}^{T} \gamma^t r_t], \tag{4.1}$$

$$\text{s.t. } \alpha_{i,j} \in [\alpha_{min}, \alpha_{max}], \tag{4.2}$$

$$\beta_{i,j} \in [\beta_{min}, \beta_{max}], \tag{4.3}$$

$$\gamma_{i,j} \in [\gamma_{min}, \gamma_{max}], \tag{4.4}$$

where γ is the discount factor, r_t is the system performance measured by multiple metrics (defined in Sec. 4.4.2), α_{min} and α_{max} represent the controlled range of AULB actions, and β_{min}, β_{max}, γ_{min} and γ_{max} define the controllable range of IULB actions.

4.3.2 System Model Learning

One of the vital components of this work is system model learning, which learns network traffic patterns in a black-box manner. In particular, we learn two system dynamics, which are the state transition function and reward function. We train a dynamic model as a teacher for each traffic pattern. Then, we distill the knowledge from multiple teachers into one student model. The student model will be used to train the policy.

The Teacher Model

In this work, we assume the state transition function to be a deterministic function of the state s_t and action a_t. In this study, we employed a neural network with three hidden layers to approximate the function, based on the outcomes of hyper-parameter tuning. This tuning process utilized a grid search strategy to systematically explore and identify the optimal hyper-parameters. For each model, we learn two dynamic functions: state transition function $f_{\phi_k^T}(s_t, a_t)$ and reward function $f_{\eta_k^T}(s_t, a_t)$, where $k \in [1, \ldots, K]$ and K is the number of teacher models. In our work, the reward function remains the same across tasks while the dynamics vary (i.e., various traffic patterns). Therefore, each teacher model constitutes a different belief about what the dynamics in the true environment could be and minimizes the loss:

$$\mathcal{L}_T = \sum_{k=1}^{K} \sum_{(s_t, a_t, s_{t+1}, r_t) \in \mathcal{D}_k} [||s_{t+1} - f_{\phi_k^T}(s_t, a_t)||_2^2 + ||r_t - f_{\eta_k^T}(s_t, a_t)||_2^2]. \tag{4.5}$$

In this thesis, we train teacher models using datasets individually collected from each base station. The training of these models is continued until each model converges to a predetermined, stable policy.

The Student Model

To distill the knowledge from the multi-teacher model, we adopt the idea from [5] and construct two parts in the student model loss function, which are ground truth loss and knowledge distillation loss, i.e.:

$$\mathcal{L}_S = \sum_{k=1}^{K} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}_k} [||s_{t+1} - f_{\phi^S}(s_t, a_t)||_2^2 + ||f_{\phi_k^T}(s_t, a_t) - f_{\phi^S}(s_t, a_t)||_2^2], \tag{4.6}$$

where f_{ϕ^S} is the student network parameterized by ϕ^S, and L_S denotes the overall loss function of the student model. The sum is taken over K, representing the total number of teacher models. $(s_t, a_t, s_{t+1}) \in D_k$ indicates that the summation is over all data points in the dataset D_k, comprising a state at time t (s_t), an action at time t (a_t), and the subsequent state at time $t+1$ (s_{t+1}). The term $||s_{t+1} - f_{\phi_S}(s_t, a_t)||_2^2$ is the ground truth loss, measuring the difference between the actual next state and the next state predicted by the student model, using squared Euclidean distance. The term $||f_{\phi_{T_k}}(s_t, a_t) - f_{\phi_S}(s_t, a_t)||_2^2$ represents the knowledge distillation loss, quantifying the difference between the predictions of the k-th teacher model and the student model, also using squared Euclidean distance.

This loss function is crucial in the knowledge distillation process, combining a ground truth loss component, which ensures accuracy in state prediction, with a knowledge distillation loss component, which aligns the student model's predictions with those of multiple teacher models.

4.3.3 Policy Rehearsal

$$\mathcal{L}_{S_{reward}} = \sum_{k=1}^{K} \sum_{(s_t,a_t,s_{t+1})\in\mathcal{D}_k} [||r_{t+1} - f_{\eta^S}(s_t,a_t)||_2^2 + ||f_{\eta_k^T}(s_t,a_t) - f_{\eta^S}(s_t,a_t)||_2^2],$$

We leverage the learned student model for policy learning. In this step, we train the policy using generated trajectories, which we call policy rehearsal. We name generated trajectories rehearsal trajectories. At each time-step t, the student model computes the hypothetical state $\hat{s}_{t+1}$ and reward $\hat{r}_t$. This step mirrors the structure of the underlying MDP model $\mathcal{M}$ and computes an approximate MDP $\hat{\mathcal{M}}$ with the expected reward and state. Given such a model, we can help the agent rehearse future actions based on the current state, which could be seen as a way of planning [6]. The goal of the agent is maximizing the future return and updating the corresponding policy with respect to the approximate MDP $\hat{\mathcal{M}}$: $\hat{\eta}(\theta;\phi_S) = \mathbb{E}_{\hat{\tau}}[\sum_{t=0}^{T} r(\hat{s}_t,a_t)]$, where $\hat{\tau} = (s_0,a_0,\ldots), s_0 \sim \rho_0(\cdot), a_t \sim \pi_\theta(\cdot|s_t)$ and $\hat{s}_{t+1} = f_{\phi^S}$.

Specifically, we adopt the PPO algorithm as our policy training strategy. In addition, we adopt the idea from [7] for early stopping. We will keep the policy learning with rehearsal trajectories over a while until $\hat{\eta}$ no longer improves.

4.3.4 Policy Evaluation

After policy rehearsal, we evaluate the learned policy in the real environment. We perform a shallow trail, which rollout the environment with fixed timesteps using the current policy, and calculate the collected return. In the real MDP $\mathcal{M}$, the shallow trail is defined as $\eta(\theta) = \mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)]$, where T is a small number of steps, and $r(s_t, a_t)$ denotes the immediate reward received at state s_t after acting action a_t. The policy learning continues as long as the current policy still improves:

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}[\eta(\theta_{new}) > \eta(\theta_{old}) + C], \tag{4.7}$$

where C is a threshold value that measures the relative improvements. We perform T times of shallow trail and compute the average value. The iteration continues as long as this ratio exceeds a certain threshold. In our implementation, we find 0.7 is a good threshold.

4.3.5 Student Model Ensemble

To prevent multi-teacher KD from unstable training performance [8], we propose a student model ensemble solution. Instead of training one single student model, we train a set of student models $f_{\phi_1^S}, \ldots, f_{\phi_k^S}$ using the same set of teacher models. All student models are trained parallel via the same loss function Eqn. 4.6. Furthermore, to enlarge the discrepancies among these student models, we uniformly sample different parameters' weights $\phi_k^S, k \in K$

Algorithm 1: Multi-Teacher Model Based Reinforcement Learning (MOBA)

Require: Inner and outer step size α, β; Initialize policy π_θ, models $f_{\phi_1}, \ldots, f_{\phi_k}$, and $\mathcal{D} = 0$ **for** *T episodes* **do**
  Sample trajectories from the real environment with policies $\pi_{\theta_1}, \ldots, \pi_{\theta_k}$;
  Add them to $\mathcal{D}_1, \ldots, \mathcal{D}_k$;
  Train model f_{ϕ_k} using $\mathcal{D}_k$;
  for *Model* $k \leftarrow 1$ *to* K **do**
    Optimise ϕ_k using Eqn. 4.5;
  while *Performance* $\hat{\eta}$ *still improves* **do**
    Training K student model ϕ_k^S using Eqn. 4.6;
    Sample imaginary trajectories from student models using Eqn. 4.8;
    Update the policy according to the imaginary trajectories;
  Terminate if the policy evaluation ratio (using Eqn. 4.7) is below the threshold;

as the model initialization.

States and Rewards Ensemble. Since our evaluation environments are continuous, it is no harm to simply average the learned states and rewards. This approach not only avoids student model overfitting but also stabilizes policy learning. We define the predicted next state as $\hat{s}_{t+1}$ and $\hat{r}_t$, and the predicted reward as $\hat{r}_t$. The averaged predictions are given as:

$$\hat{s}_{t+1} = \frac{1}{K}\sum_{k=1}^{K}[f_{\phi_k^S}(s_t, a_t)],\ \hat{r}_t = \frac{1}{K}\sum_{k=1}^{K}[f_{\eta_k^S}(s_t, a_t)]. \tag{4.8}$$

Although we don't evaluate in discrete states or rewards environments, we propose a solution to generate rehearsal trajectories. In every step, we use the majority vote to decide the next state and reward. If predictions are different, we randomly choose one of them as the next state and reward. The MOBA is described in Algorithm 1.

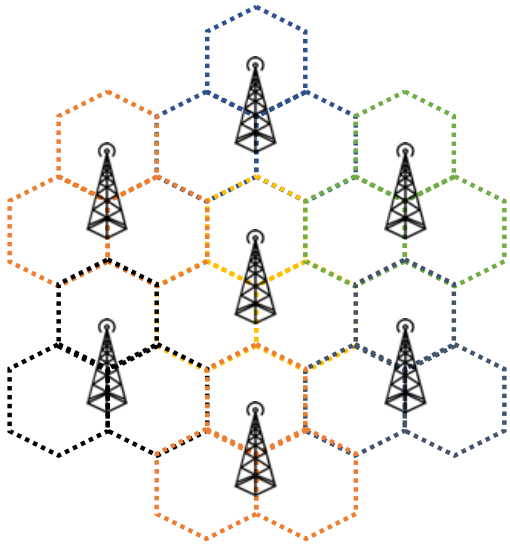
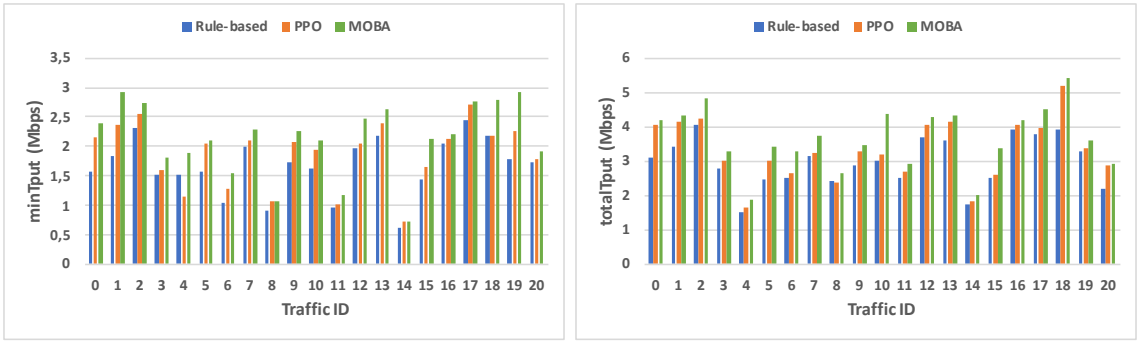


Figure 4.2: The simulation scenario contains 7 BSs. Each hexagon denotes one sector.

4.4 Evaluation on Network Load Balancing

4.4.1 Environment Setup

We evaluate our method in a communication network simulator for the load balancing task, which is illustrated in Figure 4.2. Specifically, each Base Station (BS) is made up of 3 sectors, each of which has 4 cells operating on different frequency channels. There are 26 traffic scenarios in this simulator, each of which has a different number of UEs and packet sizes.



(a) minTput **(b)** totalTput

Figure 4.3: Comparison of performance metrics (i.e., minTput and totalTpu, the higher the better) in different traffic scenarios.

4.4.2 System Performance Metrics

We define $u_{i,k}$ as the k-th UE in the i-th cell. Let $A_{i,k}$ denote the total size of packets received by UE $u_{i,k}$. We use T to represent the period of interest. In this chapter, we evaluate the system performance over the following metrics.

- **Minimum Throughput (minTput)** shows the worst-case UE performance. Maximizing this metric improves the worst-case user experience.

$$G_{minTput} = \min_{i,k} \left(\frac{A_{i,k}}{T} \right). \tag{4.9}$$

- **Total Throughput (totalTput)** evaluates overall system performance. This metric

reflects the overall provided network services.

$$G_{totalTput} = \sum_i \sum_k \frac{A_{i,k}}{T}. \tag{4.10}$$

- **Dead Cell Count (DCC)** refers to the number of cells whose throughput are less than a threshold (e.g., 0.5 Mbps in this chapter), which we treat them as dead cells. Reducing DCC can improve the utilization of different cells.

$$G_{DCC} = \frac{1}{T} \sum_k \mathbb{1}[\sum_i A_{i,k} < 0.5]. \tag{4.11}$$

- **Intensive Care Unit cell Count (ICUC)** captures the number of cells whose output are less than a threshold (e.g., 1 Mbps in this chapter). We use ICUC and DCC collaboratively measure the cells utilization.

$$G_{ICUC} = \frac{1}{T} \sum_k \mathbb{1}[\sum_i A_{i,k} < 1]. \tag{4.12}$$

4.4.3 Methods Evaluated

We compare our method with the following SOTA methods.

- **Rule-based** manages the base station load balancing using pre-programmed control parameters according to prior knowledge [1].

- **PPO** (proximal policy optimization) [9] is one of the SOTA policy update algorithms that has been widely used in control problems. To make a fair comparison, we train PPO on several representative traffic scenarios and evaluate them on other unseen scenarios.

4.4.4 Evaluation Results in Different Metrics

We first compare our method against the Rule-based method and the PPO method on the system performance of minTput and totalTpu. To select the training traffic patterns, we group the traffic scenarios into 5 clusters by using the K-nearest neighbours (KNN) method. The number of clusters is the hyper-parameters we find that show the best results. Then, for each group, we randomly select one traffic scenario as the representative. We collect trajectories from these representatives and train our teacher model accordingly.

Method	minTput (Mbps)	totalTput (Mbps)
PPO	12.5%	11.7%
MOBA	**28.6%**	**23.2%**

Table 4.1: The average relative improvement of MOBA and PPO against the Rule-based method over 21 traffic scenarios.

Figure 4.3 shows the results of minTput and totalTput. As we can see, MOBA outperforms the Rule-based method and the PPO method in all traffic scenarios. We summarize the relative improvement of MOBA and PPO against the Rule-based method in Table 4.1. The results are averaged over 21 traffic scenarios and demonstrate the

effectiveness of the proposed MOBA algorithm. The Rule-based method suffers from the worst performance because it uses fixed pre-programmed control parameters, while MOBA and PPO leverage RL to optimize control parameters. Moreover, PPO does not work as well as our MOBA because PPO is a model-free RL method that can not model the system dynamics. Specifically, in the training data, there are some similar states but with different rewards due to different traffic patterns. Without knowing the underlying system dynamics, the PPO method cannot distinguish these similar states, resulting in wrong actions and poor performance.

To overcome the aforementioned issue, MOBA adopts two mechanisms: (i) MOBA is a model-based method that models the system dynamics. When encountering similar states, the learned system model predicts the next state and potential reward, which helps the agent make the right decision. (ii) MOBA utilizes the multi-teacher KD to enhance the generalization of the trained system model. The distilled system model can predict unseen traffic patterns according to the teacher's knowledge.

Moreover, we compare our method against the Rule-based method and the PPO method on the system performance of DCC and ICUC, which can help us to understand whether the load is balanced or not. Figure 4.4 shows the results of DCC and ICUC. As we can see, compared with the Rule-based method and the PPO method, MOBA decreases both the DCC and ICUC values, which illustrates the effectiveness of our MOBA method. On the other hand, the decreased DCC and ICUC values imply that the number of working cells is

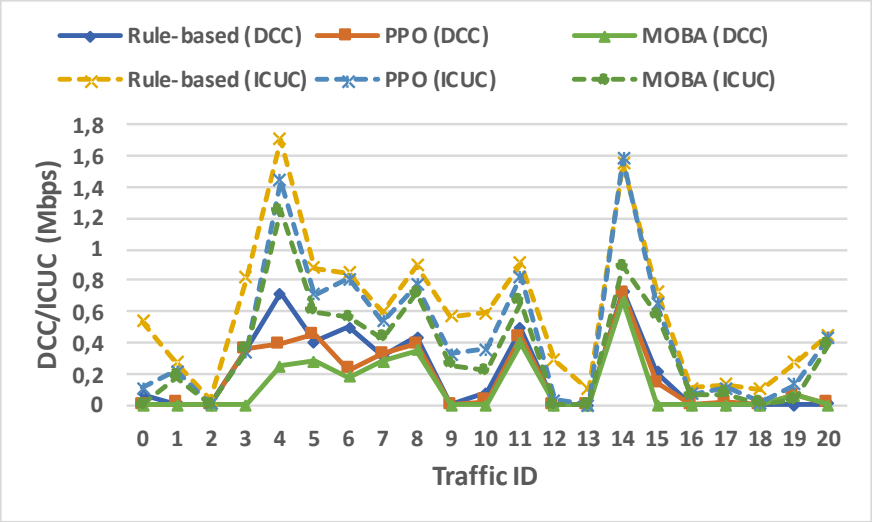


Figure 4.4: Comparison of performance metrics (i.e., DCC and ICUC, the lower the better) in different traffic scenarios.

increased. As a result, UEs can be migrated from a heavy load cell to a light load cell that was recently added/worked, i.e., the load is balanced.

4.4.5 Evaluation results in training efficiency

Lastly, we compare the training efficiency of our method and PPO. The results in Figure 4.5 show that our method can achieve a better convergence rate and result in a better performance than model-free methods (i.e., PPO). Specifically, MOBA reduce the average training time by up to 64.0%.

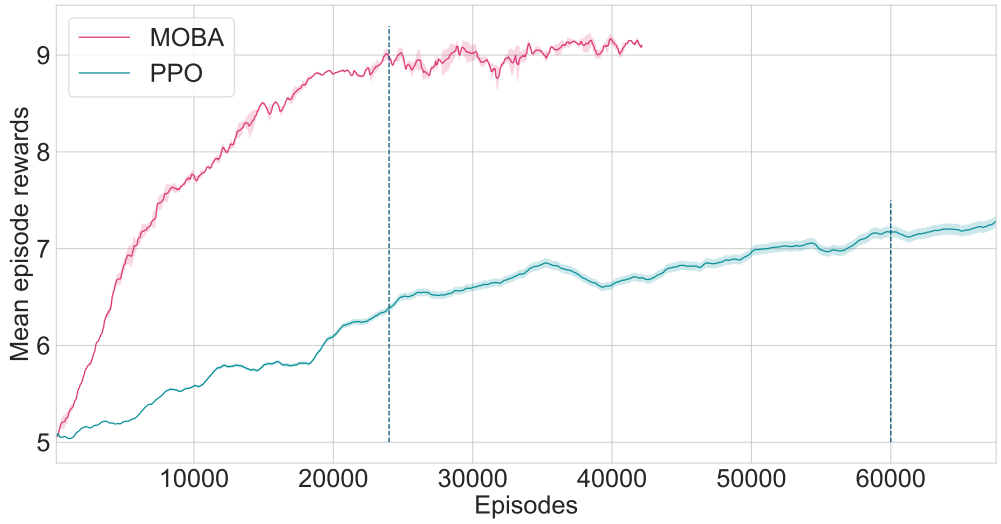


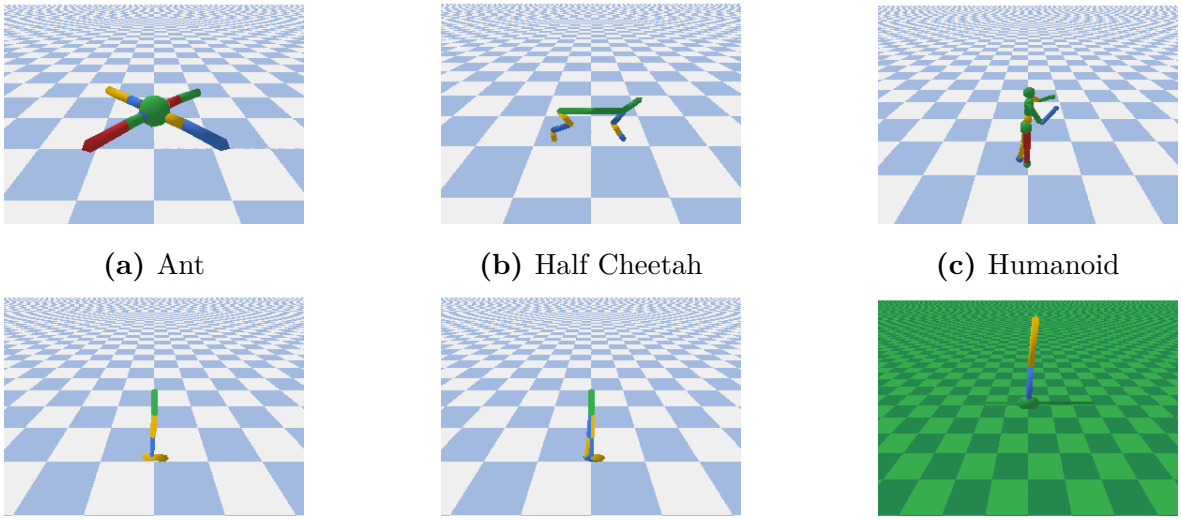
Figure 4.5: Learning curves of our method versus PPO. Each learning curve is computed in three runs with different random seeds. The y-axis is the mean episode rewards, which are the combinations of minTput and TotalTput.

4.5 Evaluation on Robotics Tasks

In this section, we further evaluate on robotics tasks to demonstrate the effectiveness of the proposed method.

4.5.1 Environment Setup

We evaluate our algorithms in continuous control benchmark tasks in PyBullet environment [10]. Specifically, we choose three standard benchmark tasks: Ant, Half Cheetah, Humanoid, Walker, Inverted Double Pendulum, and Hopper, which are shown in Figure 4.6. We create several instances, which only differ in environment parameters for each task, and each instance has a corresponding teacher model.



(a) Ant **(b)** Half Cheetah **(c)** Humanoid

(d) Hopper **(e)** Walker2D **(f)** Pendulum

Figure 4.6: The visualization of Pybullet environments

4.5.2 Comparison to SOTA: Model-Free Algorithms

We compare our method with state-of-the-art model free RL algorithms to show that our method can achieve asymptotic performance with better convergence rate. The results are shown in Figure 4.7.

- PPO: proximal policy optimization [9], which is one of the state-of-the-art policy update algorithms that has been widely used in continuous control problems .
- SAC: soft actor-critic [11] is an algorithm that optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches.
- TD3: twin delayed DDPG [12] is an algorithm that addresses overestimate Q-values

issue by introducing three critical tricks: clipped double-Q learning, “delayed” policy updates and target policy smoothing.

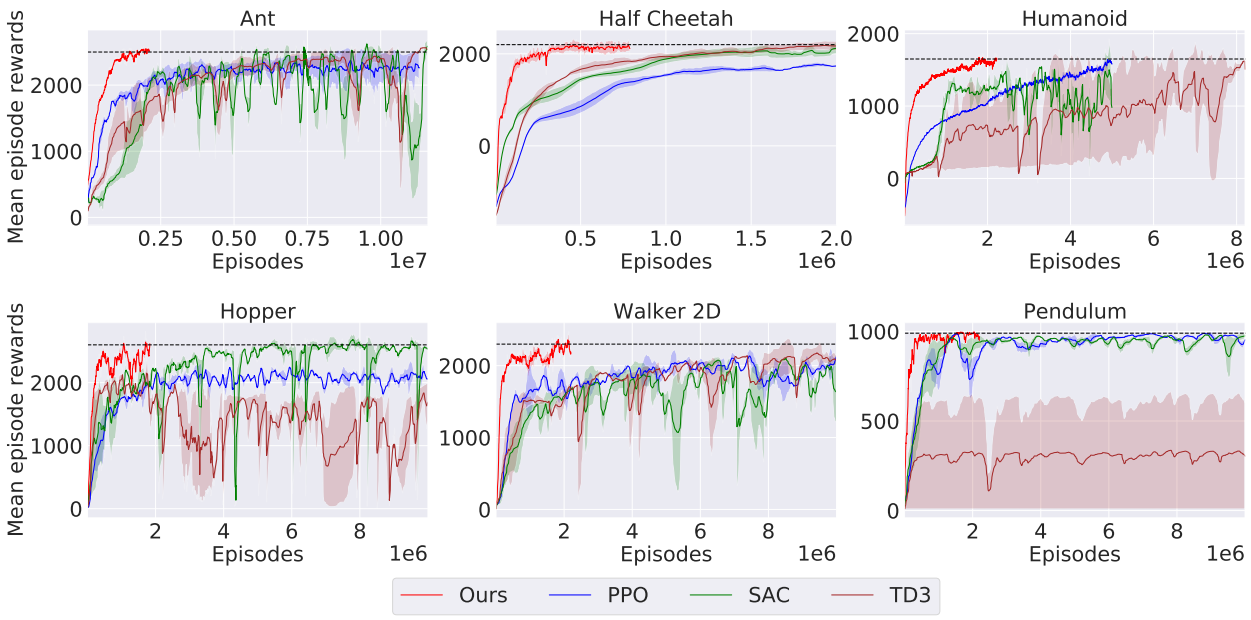


Figure 4.7: Learning curves of our method versus state-of-the-art model-free algorithms. Each learning curve is computed in threee runs with different random seeds. The dash line depict the desired best reward. MOBA (”Ours”) achieves faster convergence rate and achieves better performance than model-free methods.

In all the locomotion tasks our method MOBA can outperform other model-free methods in terms of convergence rate and mean episode return. More specifically, MOBA reduce average 64.0% of the training time in most of tasks, 75% in pendulum task, and outperform other algorihtms in walker2D. Since we adopt PPO as our policy learning algorithm, our method can effectively improve the PPO performance in challenging tasks: hopper, walker2D half cheetah, which PPO is hard to converge to the optimal solution.

4.5.3 Comparison to SOTA: Model-Based Algorithms

To further understand advantages of multi-teacher knowledge distillation in improving data-efficiency, we conduct some experiments with state-of-the-art model-based RL algorithms. The results are shown in Figure 4.8.

- GrBAL: gradient-based adaptive learning [13], which uses gradient-based method to learn a online adaptation of models in dynamic enviornmtns.
- MBPO: model-based policy optimization [14], which incorporates a linear approximation of model generalization into the analysis and justify using the model for truncated rollouts.

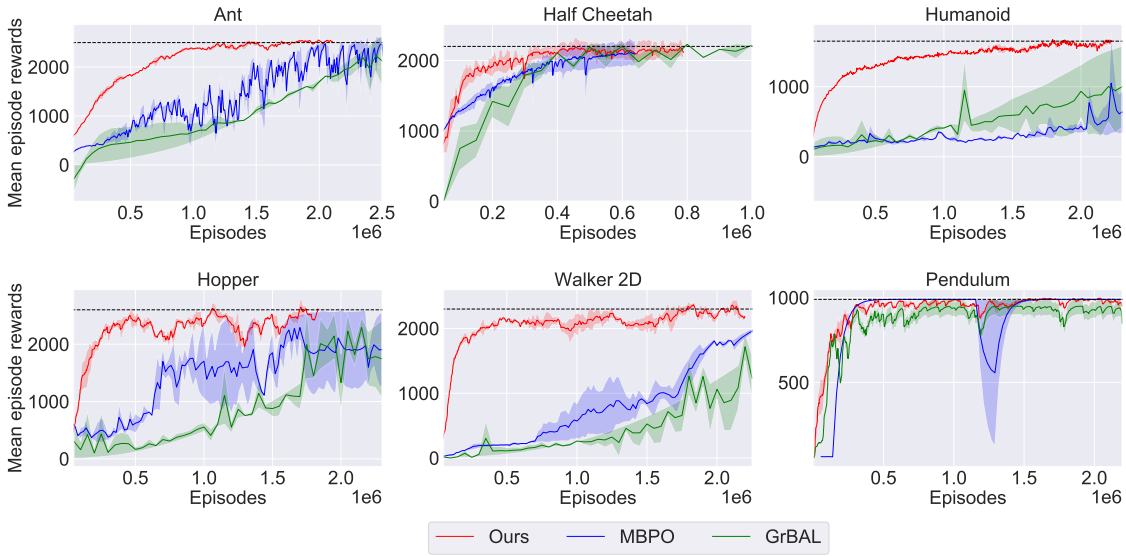


Figure 4.8: Learning curves of our method versus state-of-the-art model-base algorithms. Each learning curve is computed in three runs with different random seeds. MOBA ("Ours") achieves faster convergence rate than other model-free methods.

All these method can achieve the desired performance with less training time compared to model-based method shown in Figure 4.7. However, MOBA can still outperform other model-based method in terms of sample efficiency in all tasks. We think the reason is the policy rehearsal, which plays a role as planning in policy training. Previous model-based methods mainly focus on separating the agent training from environment. In stead of directly interact with environment, agent plays with the learned model and optimize its policy. However, in our method, we utilize the planning idea to rehearsal the agent trajectories with current policy and calculate its corresponding return. The agent learns to make the best decision by planning in the rehearsal trajectories and updating its policy. After the policy rehearsal, agent interact with real environment and collect real reward as well as new environment trajectories, which, as a result, helps improve the policy rehearsal in next iteration.

4.5.4 Dealing with The Model-bias Problem

To illustrate the model-bias problem and how it affect the policy learning, we empirically measure the policy performance under different dynamic systems with biased noise. We adopt the idea from [15], which add biased Gaussian noise $\mathcal{N}(b, 0.1^2)$ to the next state prediction, where $b \sim \sqcap(0, b_{max})$ is sampled from a uniform distribution between 0 and b_{max}. In Figure 4.9 We show the policy performance under different ranges of adding bias.

Results show that our method consistently outperform other two model-based algorithm when system model has some biased noises. Especially, when exposed to the strong bias

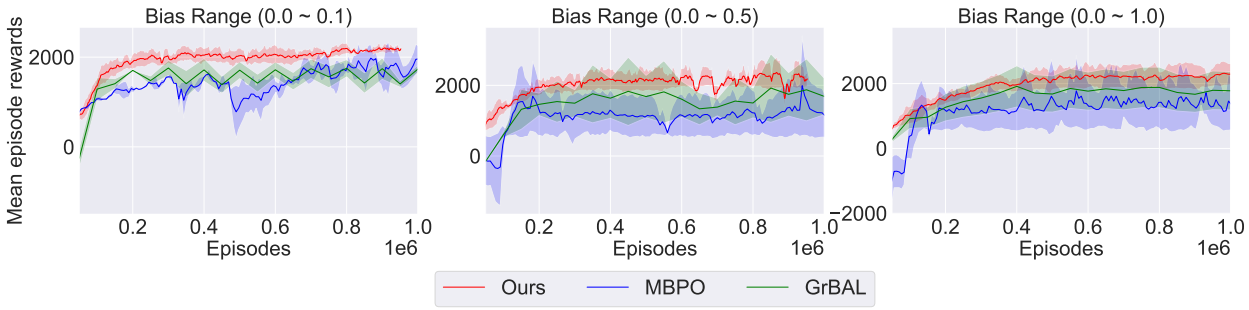


Figure 4.9: Learning curves of our methods versus SOTA model-based algorithms using three different bias ranged dynamic models in the half cheetah environment.

Method	Range (0.0∼0.1)	Range (0.0∼0.5)	Range (0.0∼1.0)
MBPO	1532(±141)	581(±369)	539(±369)
GrBAL	1650(±125)	934(±378)	1046(±370)
Ours	**1984(±60)**	**1707(±106)**	**1691(±112)**

Table 4.2: Evaluation results of different model-bias ranges

environments i.e. $b_{max} = 0.5$ and $b_{max} = 1.0$, MBPO and GrBAL show large variance in the learning curve and fail to converge to an optimal solution. On the country, MOBA manage to solve model-bias problem in all three different bias range cases, which shows its effectiveness.

4.5.5 Ablation Study

In this section, we analyze the effect of student model ensemble component. Figure 4.10 shows the results of MOBA and Vanilla-MOBA.

The results indicate that our proposed method can achieve better stability and lower variance by adding student ensemble component. This phenomenon is even more noticeable in Pendulum environment, which shows both MOBA and Vanilla-MOBA (in best case) can

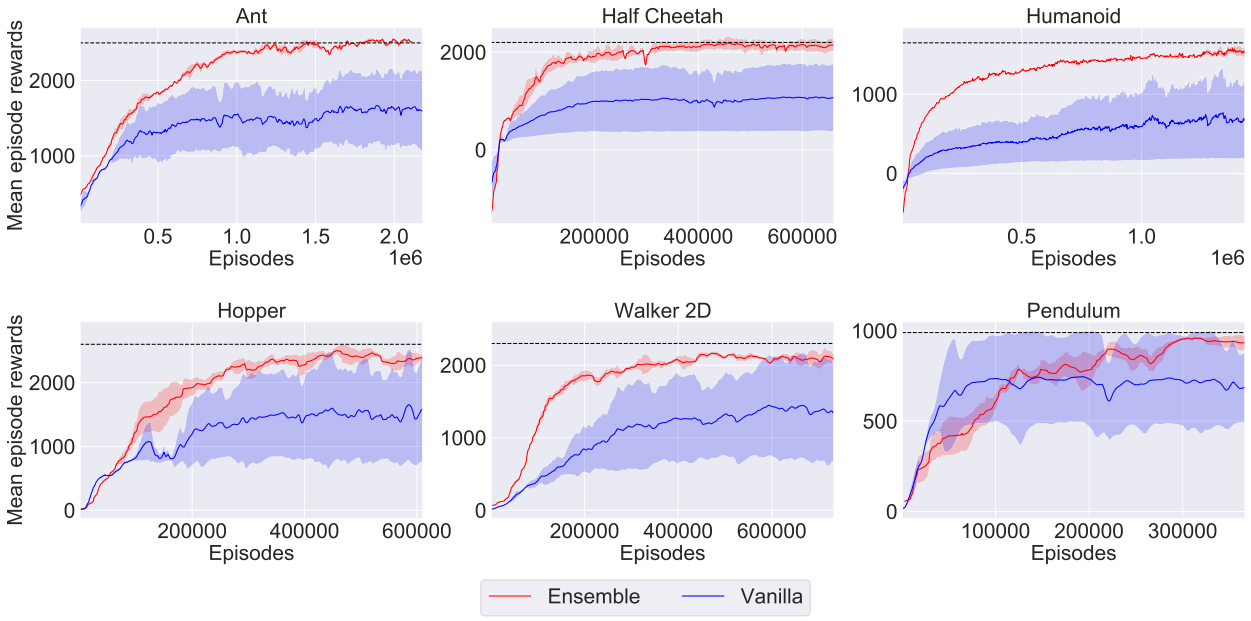


Figure 4.10: Learning curves of MOBA and Vanilla-MOBA.

find the optimal policy. However, the mean value of the Vanilla-MOBA is lower than MOBA, which shows the effectiveness of student ensemble component.

4.6 Summary

In this chapter, we propose a novel RL-based load balancing algorithm that can generalize to various traffic patterns. A major challenge lies in how to learn the underlying patterns of different scenarios and transfer them to a single model. To conquer this challenge, the proposed MOBA algorithm learns a generalized dynamic model through the knowledge distillation from multiple teacher models, where teacher models are learned from different traffic patterns. To the best of our knowledge, this is the first multi-teacher knowledge

distillation approach for network load balancing. Results demonstrate that MOBA outperforms SOTA RL-based load balancing algorithms in terms of policy generalization. MOBA increases the system's minimum and total throughput by up to 28.6% and 23.2%, respectively. Additionally, MOBA can reduce the training time by up to 64.0%.

Furthermore, evaluation results show that our method can outperform SOTA model-free and model-based methods in high-dimensional control locomotion tasks. Moreover, our method achieves the best performance when exposed to high-range model-bias environments. The ablation study shows the effectiveness of leveraging the student model ensemble component to stabilize the policy and model learning.

References

[1] D. Wu, J. Kang, Y. T. Xu, *et al.*, "Load balancing for communication networks via data-efficient deep reinforcement learning," in *GLOBECOM*, IEEE, 2021, pp. 1–7.

[2] J. Kang, X. Chen, D. Wu, *et al.*, "Hierarchical policy learning for hybrid communication load balancing," in *ICC*, IEEE, 2021, pp. 1–6.

[3] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. arXiv: `1503.02531`. [Online]. Available: `http://arxiv.org/abs/1503.02531`.

[4] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *CoRR*, vol. abs/2006.16712, 2020.

[5] S. You, C. Xu, C. Xu, and D. Tao, "Learning from multiple teacher networks," in *KDD*, ACM, 2017, pp. 1285–1294.

[6] J. Schrittwieser, I. Antonoglou, T. Hubert, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *CoRR*, vol. abs/1911.08265, 2019. arXiv: 1911.08265. [Online]. Available: http://arxiv.org/abs/1911.08265.

[7] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," in *ICLR (Poster)*, OpenReview.net, 2018.

[8] L. Wang and K. Yoon, "Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks," *CoRR*, vol. abs/2004.05937, 2020.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[10] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, http://pybullet.org, 2016.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1856–1865.

[12] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1582–1591.

[13] A. Nagabandi, I. Clavera, S. Liu, *et al.*, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. [Online]. Available: `https://openreview.net/forum?id=HyztsoC5Y7`.

[14] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 12 498–12 509. [Online]. Available: `https://proceedings.neurips.cc/paper/2019/hash/5faf461eff3099671ad63c6f3f094f7f-Abstract.html`.

[15] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, ser. Proceedings of Machine Learning Research, vol. 87, PMLR, 2018, pp. 617–629. [Online]. Available: `http://proceedings.mlr.press/v87/clavera18a.html`.

Chapter 5

Curriculum Learning Enhanced Training Efficiency

As we conclude Chapter 4, we have seen how our models and algorithms effectively address the complexities of load balancing and robotics tasks. The insights gained here lay the groundwork for further exploration into advanced methodologies. In Chapter 5, we will build upon these foundational concepts, delving into the realm of knowledge distillation and its application in machine learning frameworks.

5.1 Introduction

The concept that humans frequently organize their learning into a curriculum of interdependent processes according to their capabilities was first introduced to machine

learning in [1]. Over time, curriculum learning has become more widely used in machine learning to control the stream of examples provided to training algorithms [2], to adapt model capacity [3], and to organize exploration [4]. Automatic curriculum learning (ACL) for deep reinforcement learning (DRL) [5] has recently emerged as a promising tool to learn how to adapt a robot's learning tasks to its capabilities during training. ACL can be applied to robotics' policies learning in various ways, including adapting initial states [6], shaping reward functions [7], generating goals [8]. More broadly, curriculum learning can also be used to modify the environment itself. For example, it can add new entities to the world or change the behaviors of other agents [9].

Oftentimes, only a single ACL paradigm (*e.g.*, generating subgoals) is considered. It remains an open question whether different paradigms are complementary to each other and if yes, how to combine them in a more effective manner similar to how the "rainbow" approach of [10] has greatly improved DRL performance in Atari games. Multi-task learning is notoriously difficult, and [11] hypothesize that the optimization difficulties might be due to the gradients from different tasks conflicting with each other thus hurting the learning process. In this work, we propose a multi-task bilevel learning framework for more effective multi-objective curricula robotics' policy learning. Concretely, inspired by neural modular systems [12] and multi-task RL [13], we utilize a set of neural modules and train each of them to output a sequence of tasks with different desired goals, rewards, initial states, etc. To coordinate potentially conflicting gradients from modules in a unified parameter space,

we use a single hyper-net [14] to parameterize neural modules so that these modules generate a diverse and cooperative set of curricula. *Multi-task learning provides a natural curriculum for the hyper-net itself* since learning easier curriculum modules can be beneficial for learning more difficult curriculum modules with parameters generated by the hyper-net.

Furthermore, existing ACL methods usually rely on manually-designed paradigms of which the target and mechanism have to be clearly defined and it is therefore challenging to create a very diverse set of curriculum paradigms. Consider goal-based ACL for example, where the algorithm is tasked with learning how to rank goals to form the curriculum [15]. Many of these curriculum paradigms are based on simple intuitions that are inspired by learning in humans, but they usually take too simple forms (*e.g.*, generating subgoals) to apply to neural models. Instead, we propose to augment the hand-designed curricula introduced above with an *abstract* curriculum of which paradigm is learned from scratch. More concretely, we take the idea from memory-augmented meta-DRL [16] and equip the hyper-net with a non-parametric memory module, which is also directly connected to the DRL agent. The hyper-net can write entries to and update items in the memory, through which the DRL agent can interact with the environment under the guidance of the abstract curriculum maintained in the memory. The *write-only* permission given to the hyper-net over the memory is distinct from the common use of memory modules in meta-DRL literature, where the memories are both readable and writable. We point out that the hyper-net is instantiated as a recurrent neural network [17] which has its

internal memory mechanism and thus a write-only extra memory module is enough. Another key perspective is that *such a write-only memory module suffices to capture the essence of many curriculum paradigms*. For instance, the subgoal-based curriculum can take the form of a sequence of coordinates in a game which can be easily generated as a hyper-net and stored in the memory module.

The contributions of this work are as follows:

1. We introduce multi-objective curricula learning approach for improving the sample efficiency of solving challenging deep reinforcement learning tasks, which is an important problem that has not been adequately addressed before.

2. We further propose a unified automatic curriculum learning framework to create multi-objective but coherent curricula that are generated by a set of parametric curriculum modules. Each curriculum module is instantiated as a neural network and is responsible for generating a particular curriculum. To coordinate those potentially conflicting modules in unified parameter space, we propose a multi-task hyper-net learning framework that uses a single hyper-net to **parameterize** all those curriculum modules.

5.2 Preliminaries

Automatic curriculum learning (ACL) is a learning paradigm where an agent is trained iteratively following a curriculum to ease learning and exploration in a multi-task problem. Since it is not feasible to manually design a curriculum for each task, recent work has proposed to create an implicit curriculum directly from the task objective. Concretely, it aims to maximize a metric P computed over a set of target tasks $T \sim \mathcal{T}_{target}$ after some episodes t'. Following the notation in [5], the objective is set to: $\max_{\mathcal{D}} \int_{T \sim \mathcal{T}_{target}} P_T^{t'} dT$, where $\mathcal{D} : \mathcal{H} \rightarrow \mathcal{T}_{target}$ is a task selection function. The input $\mathcal{H}$ can be consist of any information about past interactions. For example, in our experiments, the history consists of the last state of the episode., and the output of $\mathcal{D}$ is a sequence of generated episodes, which contain different desired goals, initial states, rewards, etc.

Hyper-networks were proposed in [14] where one network (hyper-net) is used to generate the weights of another network. All the parameters of both networks are trained end-to-end using backpropagation. We follow the notation in [18] and suppose that we aim to model a target function $y : \mathcal{X} \times \mathcal{I} \rightarrow \mathbb{R}$, where $x \in \mathcal{X}$ is independent of the task and $I \in \mathcal{I}$ depends on the task. A **base** neural network $f_b(x; f_h(I; \theta_h))$ can be seen as a composite function, where $f_b : \mathcal{X} \rightarrow \mathbb{R}$ and $f_h : \mathcal{I} \rightarrow \Theta_b$. Conditioned on the task information I, the small **hyper**-net $f_h(I; \theta_h)$ generates the parameters θ_b of base-net f_b. Note that θ_b is never updated using loss gradients directly.

5.3 Learning Multi-Objective Curricula

We use a single hyper-net to dynamically parameterize all the curriculum modules over time and modify the memory module shared with the DRL agent. We call this framework a Multi-Objective Curricula (MOC). This novel design encourages different curriculum modules to merge and exchange information through the shared hyper-net.

Following the design of hyper-networks with recurrence [14], this hyper-net is instantiated as a recurrent neural network (RNN), which we refer to as the **Hyper-RNN**, denoted as $f_h(I;\theta_h)$, in the rest of this chapter to emphasize its dynamic nature. Additionally, the Hyper-RNN can be viewed as a configurator for other modules as suggested in [19]. Our motivation for the adoption of an RNN design is its capability for producing a distinct set of curricula for every episode, which strikes a better trade-off between the number of model parameters and its expressiveness. On the other hand, each manually designed curriculum module is also instantiated as an RNN, which is referred as a **Base-RNN** $f_b(x;\theta_b)$ parameterized by $\theta_b = f_h(I;\theta_h)$. Each Base-RNN is responsible for producing a specific curriculum, *e.g.*, a series of sub-goals.

The architecture of MOC-DRL is depicted in Figure 5.1, and its corresponding pseudo-code is given in Alg. 2. We formulate the training procedure as a bilevel optimization problem [20] where we minimize an outer-level objective that depends on the solution of the inner-level tasks.

In our case, the *outer*-level optimization comes from the curriculum generation loop

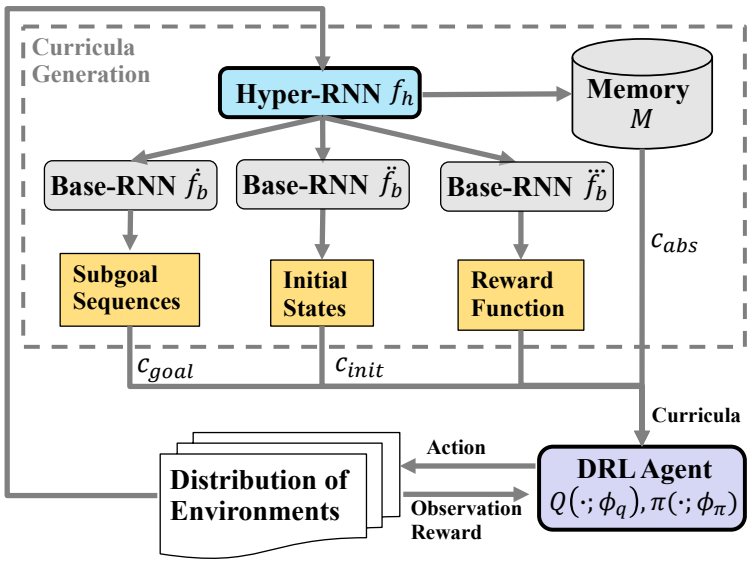


Figure 5.1: Illustration of MOC-DRL with two loops. Curricula generation corresponds to the outer-level loop. The DRL agent interacts with the environment in the inner-level loop.

where each step is an episode denoted as t'. On the other hand, the *inner*-level optimization involves a common DRL agent training loop on the interactions between the environment and the DRL agent, where each time-step at this level is denoted as t. We defer the discussion on the details to Sec. 5.3.3.

Inputs, I, of the Hyper-RNN, f_h, consist of: (1) the final state of the last episode, and (2) role identifier for each curriculum module (*e.g.*, for initial states generation) represented as a one-hot encoding. Ideally, we expect each Base-RNN to have its own particular role, which is specific to each curriculum.When generating the parameters for each Base-RNN, we additionally feed the role identifier representation to the Hyper-RNN.

Outputs of the Hyper-RNN at episode t' include: (1) parameters $\theta_b^{t'}$ for each Base-

RNN, and (2) the abstract curriculum, $\mathbf{h}_h^{t'}$, maintained in the memory module. Here $\mathbf{h}_h^{t'}$ corresponds to the hidden states of the Hyper-RNN such that $[\theta_b^{t'}, \mathbf{h}_h^{t'}] = f_h(I^{t'}; \theta_h)$.

Algorithm 2: Multi-Objective Curricula Deep Reinforcement Learning (MOC-DRL).

```
for Episode t' in 1 to T_outer do
    · Sample a new environment from the distribution of environments;
    · Hyper-RNN generates parameters for each curriculum module;
    for Base-RNN in 1 to 3 do
        · Generate a curriculum;
    · Hyper-RNN updates the abstract curriculum in the memory;
    for Training step t in 1 to T_inner do
        · DRL agent reads memory;
        · Train DRL agent following curricula;
    · Update Hyper-RNN based on outer-level objective;
```

5.3.1 Manually Designed Curricula

In Sec. 5.3.1, we describe the details of generating manually designed curricula while the process of updating the abstract curriculum is described in Sec. 5.3.2. We describe how to train them in Sec. 5.3.3.

In this work, we use three curriculum modules responsible for generating pre-defined curricula [5]: *initial state generator, sub-goal state generator,* and *reward shaping generator.*

Our approach can be easily extended to include other forms of curricula (*e.g.*, selecting environments from a discrete set [21]) by adding another curriculum generator to the shared hyper-net. These Base-RNNs simultaneously output the actual curricula for the DRL agent in a synergistic manner. It should be noted that these Base-RNNs are not directly updated by loss gradients, as their *pseudo-parameters* are generated by the Hyper-RNN.

Generating subgoal state g^t as curriculum $\mathbf{c}_{goal}$ with Base-RNN $\dot{f}_b$. As one popular choice in ACL for DRL, the subgoals can be selected from discrete sets [8] or a continuous goal space [15]. A suitable subgoal state g^t can ease the learning procedures by guiding the agent on how to achieve subgoals step by step and ultimately solving the final task.

To incorporate the subgoal state in the overall computation graph, in this chapter, we adopt the idea from universal value functions [22] and modify the action-value function, $Q(\cdot;\phi_q)$, to combine the generated subgoal state with other information $Q := Q(s^t, a^t, g^t; \phi_q) = Q(s^t, a^t, \mathbf{c}_{goal}; \phi_q)$, where s^t is the state, a^t is the action, and g^t is the generated subgoal state. The loss is defined as $\mathcal{J}_{goal} = \mathbb{E}_{(s^t,a^t,r^t,s^{t+1},g^t)\sim H_{buf}}[(Q(s^t, a^t, \mathbf{c}_{goal}; \phi_q) - \dot{y})^2]$, where $\dot{y}$ is the one-step look-ahead:

$$\dot{y} = r^t + \lambda \mathbb{E}_{a^{t+1}\sim\pi_\theta(s^{t+1})}[Q(s^t, a^t, \mathbf{c}_{goal}; \phi_q) \quad - \log(\pi(a^{t+1}|s^{t+1}; \phi_\pi))], \tag{5.1}$$

H_{buf} is the replay buffer and λ is the discount factor.

Generating initial state $\mathbf{s}_0$ as curriculum $\mathbf{c}_{init}$ with Base-RNN $\ddot{f}_b$. Intuitively, if the starting state $\mathbf{s_0}$ for the agent is close to the end-goal state, the training would become easier, which forms a natural curriculum for training tasks whose difficulty depends on a proper distance between the initial state and the end-goal state. This method has been shown effective in control tasks with sparse rewards [6], [23]. To simplify implementation, even though we only need a single initial state $\mathbf{s}_0$ which is independent of time, we still use a Base-RNN, $\ddot{f}_b$, to output it.

The loss for this module is: $\mathcal{J}_{init} = \mathbb{E}_{(s^t,a^t)\sim H_{buf}}[(Q(s^t, a^t, c_{init}; \phi_q) - \dot{y})^2]$, where $\dot{y}$ is defined in Eqn. 5.1.

Generating potential-based shaping function as curriculum $\mathbf{c}_{rew}$ with Base-RNN $\dddot{f}_b$. Motivated by the success of using reward shaping for scaling RL methods to handle complex domains [24], we introduce reward shaping as the third manually selected curriculum. The reward shaping function can take the form of: $\dddot{f}'_b(s^t, a^t, s^{t+1}) = \mu \cdot \dddot{f}_b(s^{t+1}) - \dddot{f}_b(s^t)$, where μ is a hyper-parameter and $\dddot{f}_b()$ is base-RNN that maps the current state with a reward. In this chapter, we add the shaping reward $\dddot{f}'_b(s^t, a^t, s^{t+1})$ to the original environment reward r. We further normalize the shaping reward between 0 and 1 to deal with wide ranges.

Following the optimal policy invariant theorem [24], we modify the look-ahead function: $\ddot{y} = r^t + \dddot{f}_b(s^t, a^t, s^{t+1} + \lambda \mathbb{E}_{a^{t+1}\sim\pi_\theta(s^{t+1})}[Q(s^t, a^t, \mathbf{c}_{rew}; \phi_q) - \log(\pi(a^{t+1}|s^{t+1}; \phi_\pi))]$. Thus the loss is defined as: $\mathcal{J}_{reward} = \mathbb{E}_{s^t,a^t,s^{t+1},a^{t+1}\sim H_{buf}}[(Q(s^t, a^t, \mathbf{c}_{rew}; \phi_q) - \ddot{y})^2]$.

We implement the reward shaping mechanism through a potential-based shaping function, specifically designed to enhance the learning process. The shaping function is defined as:

$$f'_b(s_t, a_t, s_{t+1}) = \mu \cdot f_b(s_{t+1}) - f_b(s_t) \tag{5.2}$$

where μ is a hyper-parameter and $f_b()$ represents the base-RNN mapping the current state to a reward. This shaping reward is then integrated with the original environment reward r, and normalized between 0 and 1 to accommodate a wide range of values.

5.3.2 Abstract Curriculum with Memory Mechanism

Although the aforementioned hand-designed curricula are generic enough to be applied in any environment/task, it is still limited by the number of such predefined curricula. It is reasonable to conjecture that there exist other curriculum paradigms, which might be difficult to hand-design based on human intuition. As a result, instead of solely asking the hyper-net to generate human-engineered curricula, we equip the hyper-nets with an external memory, in which the hyper-nets could read and update the memory's entries. The Hyper-RNN together with the memory can also be seen as an instantiation of shared workspaces [25] for different curricula modules, in which those modules exchange information.

By design, the content in the memory can serve as abstract curricula for the DRL agent, which is generated and adapted according to the task distribution and the agent's dynamic capacity during training. Even though there is no constraint on how exactly the hyper-net

learns to use the memory, we observe that (see Sec. 5.4.3): 1) The hyper-net can receive reliable training signals from the manually designed curriculum learning objectives[1]; 2) Using the memory module alone would result in unstable training; 3) Utilizing both the memory and manual curricula achieves the best performance and stable training. Thus, training this memory module with other manually designed curriculum modules contributes to the shaping of the content that can be stored in the memory and is beneficial for overall performance.

Specifically, external memory is updated by the Hyper-RNN. To capture the latent curriculum information, we design a neural memory mechanism similar to [26]. The form of memory is defined as a matrix M. At each episode t', the Hyper-RNN emits two vectors $\mathbf{m}_e^{t'}$, and $\mathbf{m}_a^{t'}$ as $[\mathbf{m}_e^{t'}, \mathbf{m}_a^{t'}]^T = [\sigma, \texttt{tanh}]^T(\mathbf{W}_h^{t'}\mathbf{h}_h^{t'})$: where $\mathbf{W}_h^{t'}$ is the weight matrix of Hyper-RNN to transform its internal state $\mathbf{h}_h^{t'}$ and $[\cdot]$ denotes matrix transpose. Note that $\mathbf{W}_h$ are part of the Hyper-LSTM parameters θ_h.

The Hyper-RNN *writes* the abstract curriculum into the memory, and the DRL agent can *read* the abstract curriculum information freely.

Reading. The DRL agent can read the abstract curriculum $\mathbf{c}_{abs}$ from the memory $\mathbf{M}$. The read operation is defined as: $\mathbf{c}_{abs}^{t'} = \alpha^{t'}\mathbf{M}^{t'-1}$, where $\alpha^{t'} \in \mathbb{R}^K$ represents an attention distribution over the set of entries of memory $\mathbf{M}^{t'-1}$. Each scalar element $\alpha^{t',k}$ in an attention distribution $\alpha^{t'}$ can be calculated as: $\alpha^{t',k} = \texttt{softmax}(\texttt{cosine}(\mathbf{M}^{t'-1,k}, \mathbf{m}_a^{t'-1}))$, where we choose $\texttt{cosine}(\cdot,\cdot)$ as the align function, $\mathbf{M}^{t'-1,k}$ represents the k-th row memory

[1]To some extent, tasking the hyper-net to train manually designed curriculum modules can be seen as a curriculum itself for training the abstract curriculum memory module.

vector, and $\mathbf{m}_a^{t'} \in \mathbb{R}^M$ is a add vector emitted by Hyper-RNN.

Updating. The Hyper-RNN can write and update abstract curriculum in the memory module. The write operation is performed as: $\mathbf{M}^{t'} = \mathbf{M}^{t'-1}(1 - \alpha^{t'}\mathbf{m}_e^{t'}) + \alpha^{t'}\mathbf{m}_a^{t'}$, where $\mathbf{m}_e^{t'} \in \mathbb{R}^M$ corresponds to the extent to which the current contents in the memory should be deleted.

Equipped with the above memory mechanism, the DRL learning algorithm can read the memory and utilize the retrieved information for policy learning. We incorporate the abstract curriculum into the value function by $Q(s^t, a^t, g^t, c_{abs}^{t'}; \phi_q)$. Similar to manually designed curricula, we minimize the Bellman error and define the loss function for the abstract curriculum as: $\mathcal{J}_{abstract} = \mathbb{E}_{(s^t, a^t, r^t, s^{t+1}, c_{abs}^{t'}) \sim H_{buf}}[(Q(s^t, a^t, c_{abs}^{t'}; \phi_q) - \dot{y})^2]$, where $\dot{y}$ is defined in Eqn. 5.1.

5.3.3 Bilevel Training of Hyper-RNN

After introducing the manually designed curricula in Sec. 5.3.1 and the abstract curriculum in Sec. 5.3.2, here we describe how we update the Hyper-RNN's parameters θ_h, the parameters associated with the DRL agent ϕ_q and ϕ_π. Since the Hyper-RNN's objective is to serve the DRL agent, we naturally formulate this task as a bilevel problem [20] of optimizing the parameters associated with multi-objective curricula generation by nesting one inner-level loop in an outer-level training loop.

Outer-level training of Hyper-RNN. Specifically, the inner-level loop for the DRL

agent learning and the outer-level loop for training Hyper-RNN with hyper-gradients. The outer-level loss is defined as :$\mathcal{J}_{outer} = \mathcal{J}_{initial} + \mathcal{J}_{goal} + \mathcal{J}_{reward} + \mathcal{J}_{abs}$.

Since the manually designed curricula and abstract curricula are all defined in terms of Q-function, for the implementation simplicity, we combine them together $\mathcal{J}_{outer} = \mathbb{E}_{s^t,a^t,s^{t+1},a^{t+1}\sim H_{buf}}[(Q(s^t, a^t, \mathbf{c}_{goal}, \mathbf{c}_{rew}, \mathbf{c}_{init}, \mathbf{c}_{abs}; \phi_q) - \ddot{y})^2]$. Following the formulation and implementation in [20], we obtain $\theta_h^* = \text{argmin}\,(\theta_h; \mathcal{J}_{outer}\,(\text{argmin}\,(\phi; \mathcal{J}_{inner}(\theta_h, \phi))))$.

Inner-level training of DRL agent. The parameters associated with the inner-level training, ϕ_q and ϕ_π, can be updated based on any RL algorithm. In this chapter, we use Proximal Policy Optimization (PPO) [27] which is a popular policy gradient algorithm that learns a stochastic policy.

Influence of Reward Shaping on the Learning Process The integration of reward shaping significantly alters the learning dynamics. By following the optimal policy invariant theorem, we modify the look-ahead function in our learning algorithm. This modification ensures that the policy optimization process is more aligned with both the immediate and shaped rewards, enhancing the agent's decision-making capabilities.

5.4 Experiments

We evaluate and analyze our proposed MOC DRL on the CausalWorld [28], as this environment enables us to easily design and test different types of curricula in a

fine-grained manner. This environment also provides wrapper makes the environment to execute actions on the real robot, which can be used in sim2real experiments. It should be noted that we do not utilize any causal elements of the environment. It is straightforward to apply our method to other DRL environments without major modification. Moreover, the training and evaluation task distributions are handled by CausalWorld. Take task "Pushing" as an example: for each outer loop, we use CausalWorld to generate a task with randomly sampled new goal shapes from a goal shape family

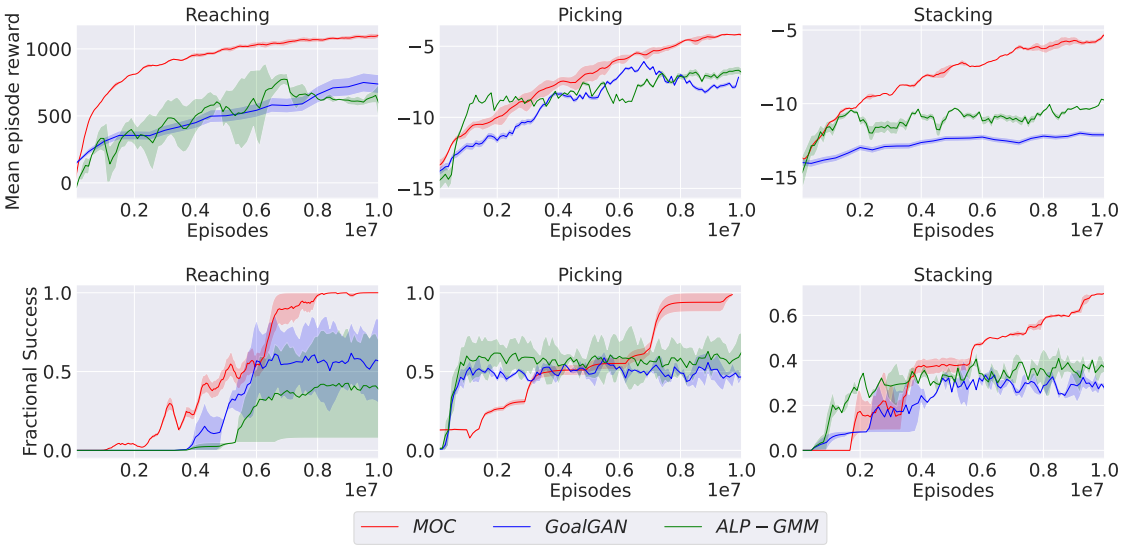


Figure 5.2: Comparisons with state-of-the-art ACL algorithms. Each learning curve is computed in three runs with different random seeds.

5.4.1 Environment Settings

We choose five out of the nine tasks introduced in CausalWorld since the other four tasks have limited support for configuring the initial and goal states. Specifically, we enumerate

these five tasks here: (1) `Reaching` requires moving a robotic arm to a goal position and reach a goal block; (2) `Pushing` requires pushing one block towards a goal position with a specific orientation (restricted to goals on the floor level); (3) `Picking` requires picking one block at a goal height above the center of the arena (restricted to goals above the floor level); (4) `Pick And Place` is an arena is divided by a fixed long block and the goal is to pick one block from one side of the arena to a goal position with a variable orientation on the other side of the fixed block; (5) `Stacking` requires stacking two blocks above each other in a specific goal position and orientation.

CausalWorld allows us to easily modify the initial states and goal states. In general, the initial state is the cylindrical position and Euler orientation of the block and goal state is the position variables of the goal block. These two control variables are both three-dimensional vectors with a fixed manipulation range. To match the range of each vector, we re-scale the generated initial states.

The reward function defined in CausalWorld is uniform across all possible goal shapes as the fractional volumetric overlap of the blocks with the goal shape, which ranges between 0 (no overlap) and 1 (complete overlap). We also re-scale the shaping reward to match this range.

We choose the PPO algorithm as our vanilla DRL policy learning method. We list the important hyper-parameters in Table. 5.1. We also provide the complete code in the supplementary material.

Parameter	Value
Discount factor (γ)	0.9995
n_steps	5000
Entropy coefficiency	0
Learning rate	0.00025
Maximum gradient norm	10
Value coefficiency	0.5
Experience buffer size	1e6
Minibatch size	128
clip parameter (ϵ)	0.3
Activation function	ReLU
Optimizer	Adam

Table 5.1: Hyper-parameter values for PPO training

	Mean Episode Reward	Success Ratio
$MOC_{RandInitState}$	936.9 (±35)	91% (±0.5%)
$MOC_{FixInitState}$	879.3 (±9)	89% (±1.1%)
$MOC_{RandGoalState}$	921.0 (±46)	91% (±0.5%)
MOC (Initial State)	1273 (±11)	100% (±0%)

(a) Analysis of initial state curriculum

	Mean Episode Reward	Success Ratio
GoalGAN	609 (±23)	56% (±18%)
ALP-GMM	568 (±26)	39% (±28%)
MOC (Goal State)	714 (±14)	68% (±15%)

(b) Analysis of subgoal curriculum

Table 5.2: Analysis of initial state curriculum and subgoal state curriculum.

5.4.2 Comparing MOC with state-of-the-art ACL methods

We compare our proposed approach with the other state-of-the-art ACL methods: (1) GoalGAN [29], which uses a generative adversarial neural network (GAN) to propose tasks for the agent to finish; (2) ALP-GMM [5], which models the agent absolute learning progress with Gaussian mixture models. None of these baselines utilize multiple curricula.

Figure 5.2 shows that MOC outperforms other ACL approaches in terms of mean episode reward, fractional success, and sample efficiency. Especially, MOC increases fractional success by up to 56.2% in all of three tasks, which illustrates the effectiveness of combining multiple curricula in a synergistic manner.

5.4.3 Ablation Study

Our proposed MOC framework consists of three key parts: the Hyper-RNN trained with hyper-gradients, multi-objective curriculum modules, and the abstract memory module. To get a better insight into MOC, we conduct an in-depth ablation study on probing these components. We first describe the MOC variants used in this section for comparison as follows: (1) $\mathbf{MOC}_{Base^-}$: MOC has the Hyper-RNN and the memory module but does not have the Base-RNNs for manually designed curricula. (2) $\mathbf{MOC}_{Memory^-}$: MOC has the Hyper-RNN to generate three curriculum modules but does not have the memory module. (3) $\mathbf{MOC}_{Memory^-,Hyper^-}$: MOC has Base-RNNs but does not have memory and Hyper-RNN components. It independently generates manually designed curricula. (4)

MOC$_{Memory^-,Goal^+}$: MOC with Hyper-RNN and one Base-RNN, but without the memory module. It only generates the subgoal curriculum as our pilot experiments show that it is consistently better than the other two manually designed curricula and is easier to analyze its behavior by visualizing the state visitation.

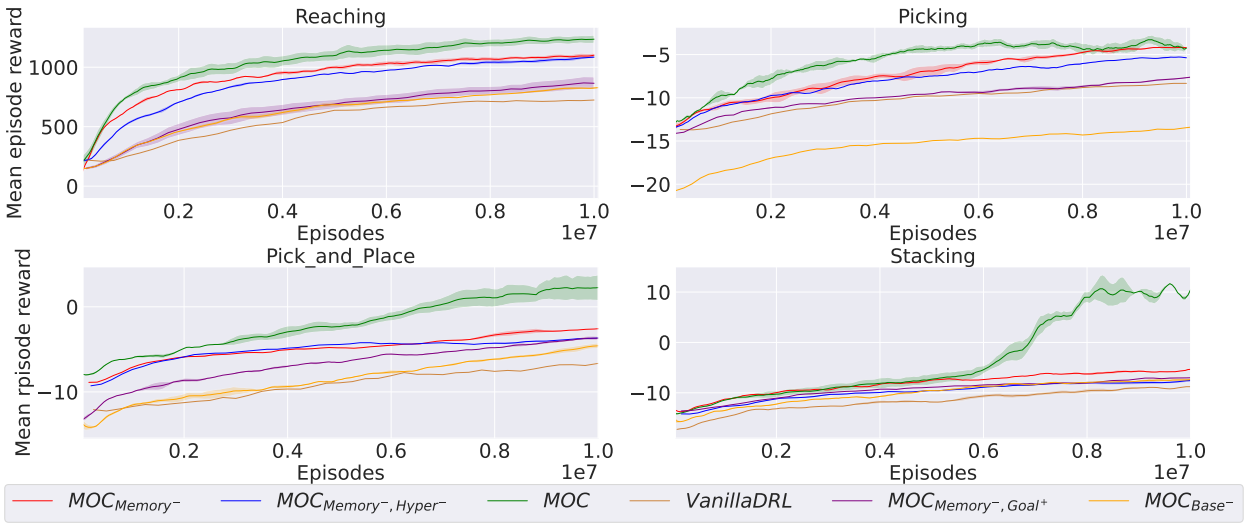


Figure 5.3: Comparison of algorithms with and without memory component on all four tasks. Each learning curve is obtained by three independent runs with different random seeds.

Ablations of Hyper-RNN. By comparing MOC$_{Memory^-}$ with MOC$_{Memory^-,Hyper^-}$ as shown in Figure 5.3, we can observe that letting a Hyper-RNN generate the parameters of different curriculum modules indeed helps in improving the sample efficiency and final performance. The advantage is even more obvious in the harder tasks `pick and place` and `stacking`. The poor performance of MOC$_{Memory^-,Hyper^-}$ may be caused by the potential conflicts among the designed curricula. For example, without coordination between the

initial state curriculum and the goal curriculum, the initial state generator may set an initial state close to the goal state, which is easy to achieve by an agent but too trivial to provide useful training information to achieve the final goal. In sharp contrast, the Hyper-RNN can solve the potential conflicts from different curricula. All the curriculum modules are dynamically generated by the same hyper-net, and there exists an implicit information sharing between the initial state and the goal state curriculum generator.

Ablations of the memory module. We aim to provide an empirical justification for the use of the memory module and its associated abstract curriculum. By comparing MOC with MOC_{Memory^-} as shown in Figure 5.3, we can see that the memory module is crucial for MOC to improve sample efficiency and final performance. Noticeably, in `pick and place` and `stacking`, we see that MOC gains a significant improvement due to the incorporation of the abstract curriculum. We expect that the abstract curriculum could provide the agent with an extra implicit curriculum that is complementary to the manually designed curricula. We also find that it is better for the Hyper-RNN to learn the abstract curriculum while generating other manually designed curricula. Learning multiple manually designed curricula provides a natural curriculum for the Hyper-RNN itself since learning easier curriculum modules can be beneficial for learning of more difficult curriculum modules with parameters generated by the Hyper-RNN.

Ablations of individual curricula. We now investigate how gradually adding more curricula affects the training of DRL agents. By comparing $\text{MOC}_{Memory^-,Goal^+}$ and

MOC_{Memory^-} as shown in Figure 5.3, we observe that training an agent with a single curriculum receives less environmental rewards as compared to the ones based on multiple curricula. This suggests that the set of generated curricula indeed helps the agent to reach intermediate states that are aligned with each other and also guides the agent to the final goal state.

5.4.4 Curricula Analysis and Visualization

In this section, we analyze the initial state curriculum and goal state curriculum. First, we replace the initial state curriculum with two different alternatives: (1) $MOC_{RandInitState}$, in which we replace the initial state curriculum in MOC with a uniformly chosen state. Other MOC components remains the same; (2) $MOC_{FixInitState}$, in which we replace the initial state curriculum in MOC with a fixed initial state. The other MOC components remains the same. (3) $MOC_{RandGoalState}$, in which we replace the goal state curriculum in MOC with a uniformly chosen state. The other MOC components remains the same. The evaluations are conducted on the `reaching` task and the results are shown in Table 5.2a. From this table, we observe that MOC with initial state curriculum outperforms other two baseline schemes in terms of mean episode rewards and success ratio. This demonstrates the effectiveness of providing initial state curriculum. Besides, since "random sampling" outperforms "fixed initial state", we conjecture that it is better to provide different initial states, which might be beneficial for exploration.

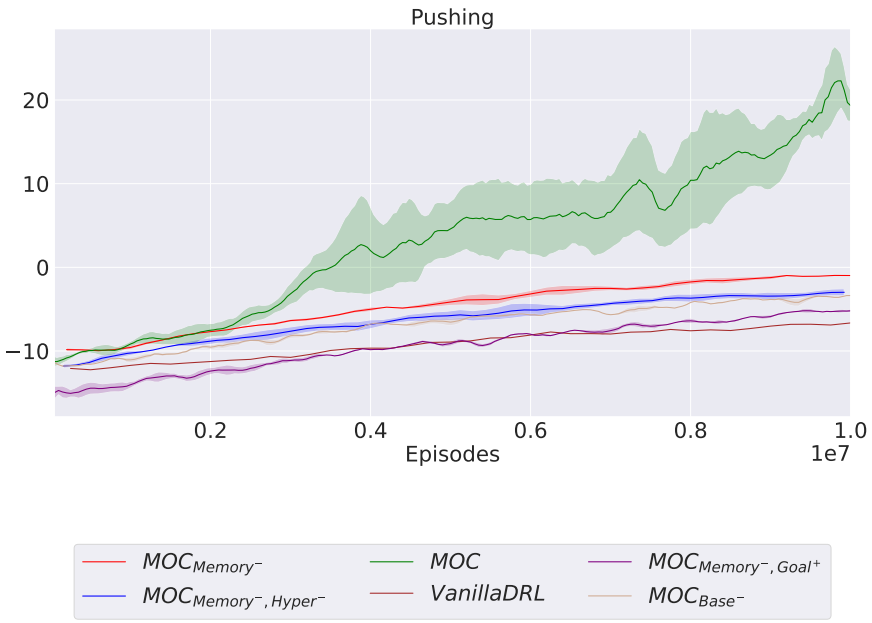


Figure 5.4: Comparison of algorithms with and without memory component in *pushing*. Each learning curve is computed in three runs with different random seeds.

In Sec. 5.4.2, we show that providing multi-objective curricula can improve the training of DRL agents. To further evaluate the advantages of hyper-RNN base-RNN framework, we conduct an experiment with GoalGAN, ALP-GMM and MOC with goal curriculum only. We evaluate on `reaching` task and the results are shown in Tab. 5.2b. In this table, we see that MOC Goal State ($\text{MOC}_{Memory^-,Goal^+}$), which is MOC has goal curriculum but doesn't have memory component, slightly outperform other two baseline schemes.

5.4.5 Additional Experimental Results

This section serves as a supplementary results for Sec. 5.4.

Figure 5.4 shows the results of with and without Hyper-RNN in `pushing` tasks. The

results validate the effectiveness of using Hyper-RNN. It is clear that, the incorporation of memory module consistently helps the DRL agent outperform other strong baselines in all scenarios. More importantly, in `pushing` task, we can observe a 5-fold improvement compared to the method with only the Hyper-RNN component.

Figure 5.4, 5.5, 5.6, 5.7 clearly validate the effectiveness of our proposed method in achieving both the best final performance and improving sample efficiency.

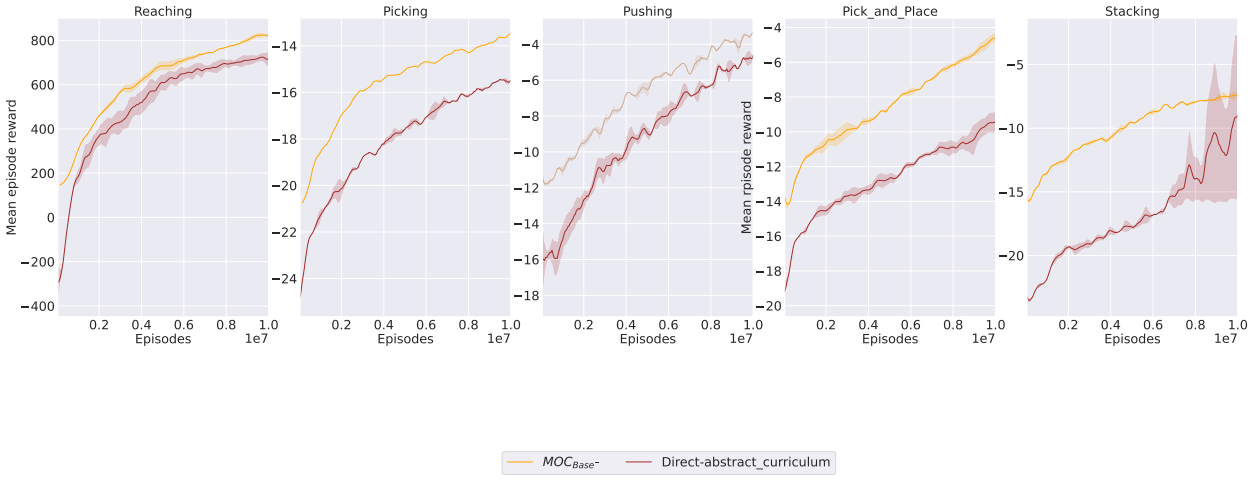


Figure 5.5: Comparison between read memory from memory and direct generate abstract curriculum

In Sec. 5.4.2, we compared MOC with state-of-the art ACL algorithms. Here, we add two more baselines algorithms. The results are shown in Figure 5.8:

- InitailGAN [6]: which generates adapting initial states for the agent to start with.
- PPO_{Reward^+}: which is a DRL agent trained with PPO algorithm and reward shaping. The shaping function is instantiated as a deep neural network.

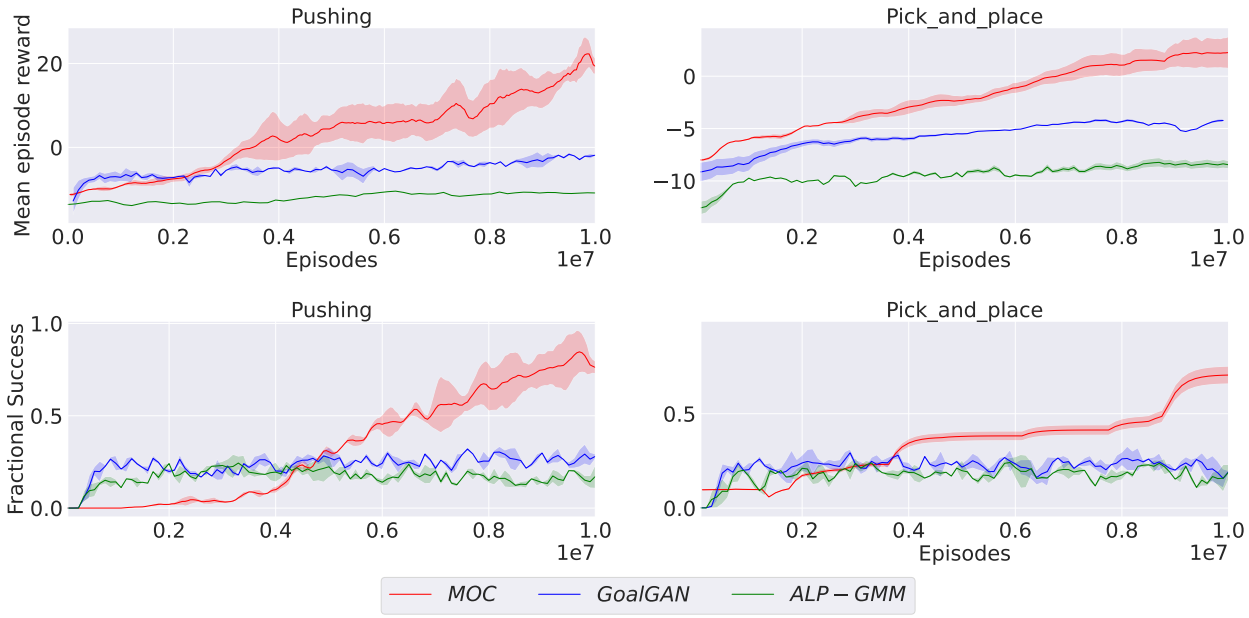


Figure 5.6: Comparison with ACL algorithms. Each learning curve is computed in three runs with different random seeds.

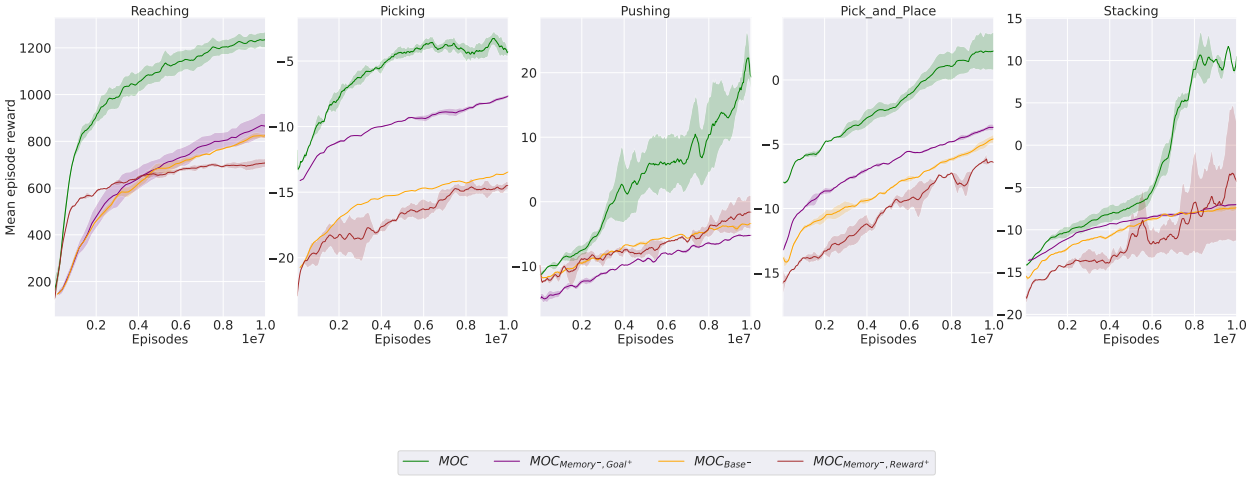


Figure 5.7: Comparison with reward curriculum only.

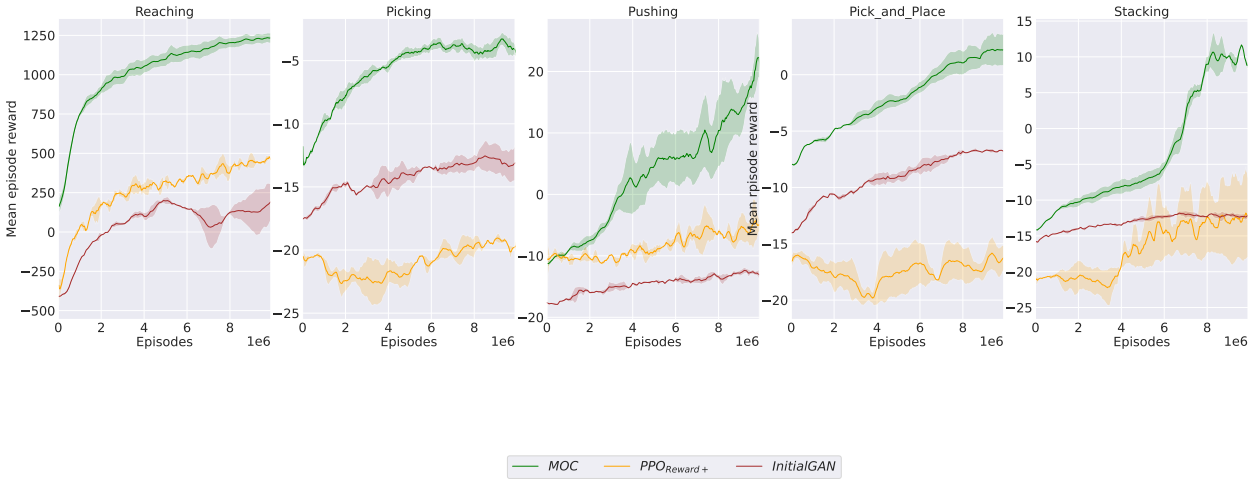
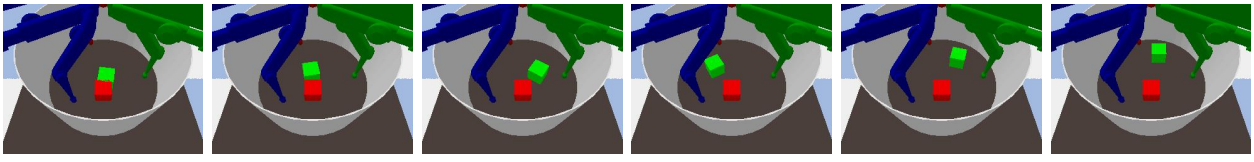


Figure 5.8: Comparison with Initial GAN and PPO with reward shaping only.

5.4.6 The visualization of generated sub-goal

The visualization of generated sub-goal state is shown in Figure 5.9. Specifically, the arm is tasked to manipulate the red cube to the position shown as a green cube. As we can see, MOC generates subgoals that gradually change from "easy" (which are close to the initial state) to "hard" (which are close to the goal state). The generated subgoals have different configurations (*e.g.*, the green cube is headed north-west in 7000k steps but is headed north-east in 9000k steps), which requires the agent to learn to delicately manipulate robot arm.



(a) 1000k steps **(b)** 3000k steps **(c)** 5000k steps **(d)** 7000k steps **(e)** 9000k steps **(f)** Goal state

Figure 5.9: Visualization of generated subgoals

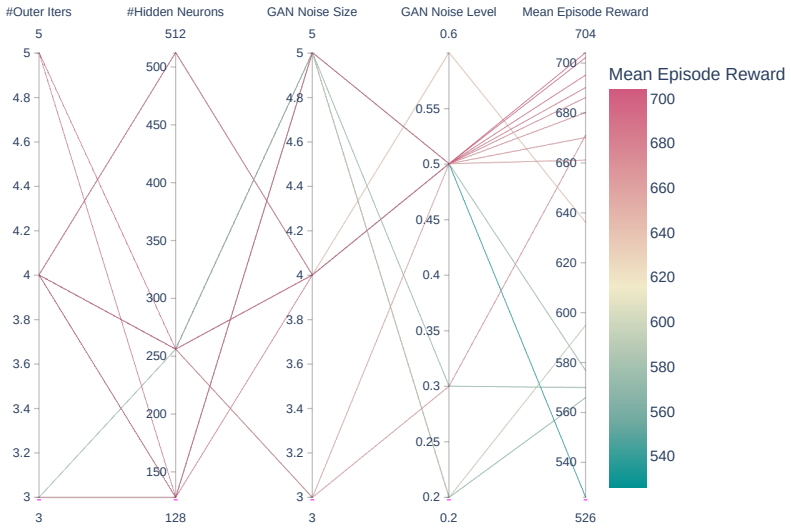
5.4.7 Hyperparameters

In this section, we extensively evaluate the influence of different hyperparameters for the baselines and MOC, where the search is done with random search. We choose the reaching and stacking tasks, which are shown in Figure 5.10, 5.11, 5.12. For example, in Figure 5.10-(a), the first column represents the different values for outer iterations. A particular horizontal line, *e.g.*, $\{4, 512, 5, 0.5\}$, indicates a particular set of hyperparameters for one experiment. Besides, during the training phase, we adopt hyperparameters of PPO from stable-baselines3 and search two hyperparameters to test the MOC sensitivity.

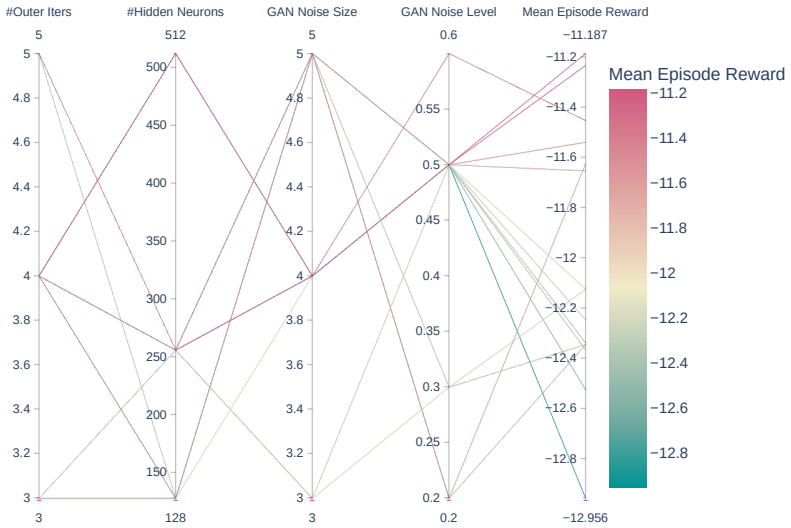
We can observe that: (1) It is clear that MOC outperforms all the baselines with extensive hyperparameter search. (2) MOC is not sensitive to different hyperparameters.

5.5 Summary

This chapter presents a multi-objective curricula learning approach for solving challenging deep robotics tasks. Our method trains a hyper-network for parameterizing multiple curriculum modules, which control the generation of initial states, subgoals, and shaping rewards. We further design a flexible memory mechanism to learn abstract curricula. Extensive experimental results demonstrate that our proposed approach significantly outperforms other state-of-the-art ACL methods in terms of sample efficiency and final performance.

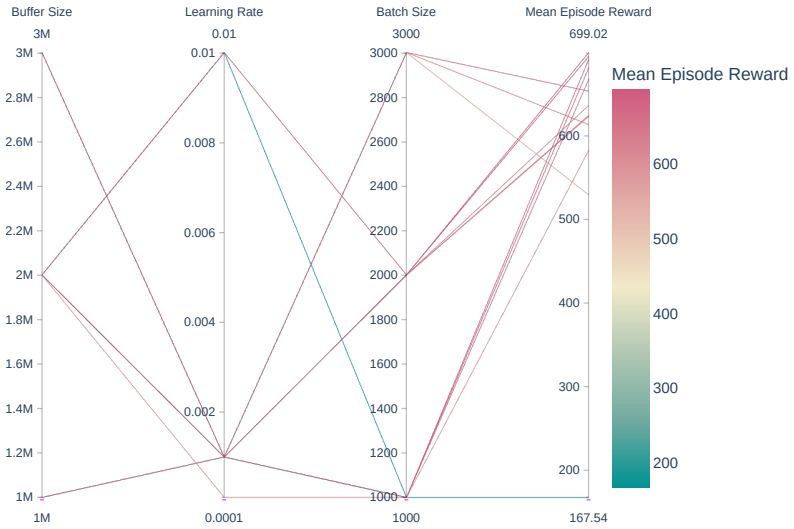


(a) Hyperparameter tuning in reaching task.

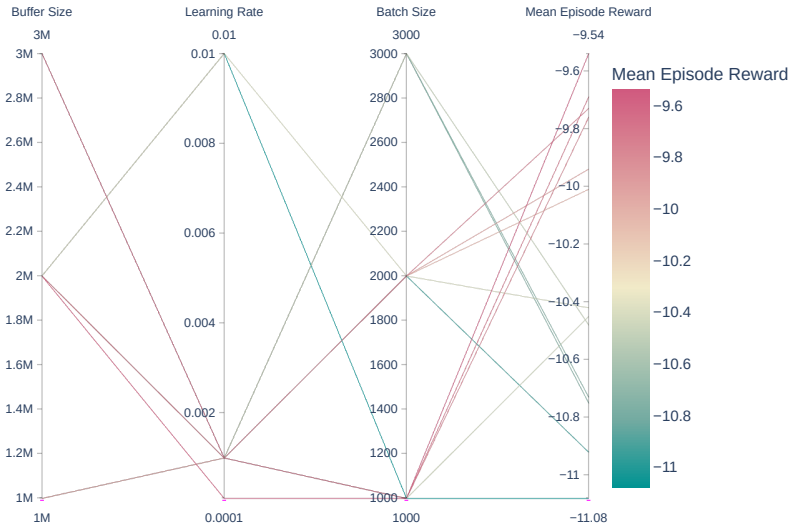


(b) Hyperparameter tuning in stacking task.

Figure 5.10: Hyperparameter tuning results for GoalGAN

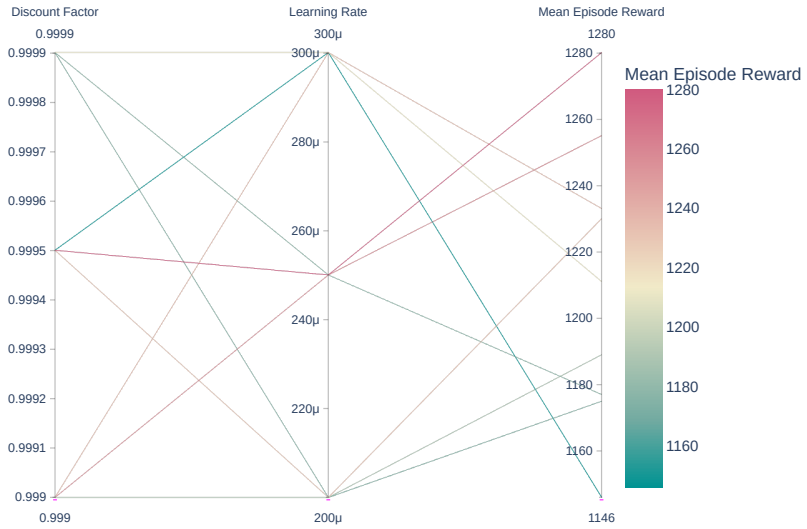


(a) Hyperparameter tuning in reaching task.

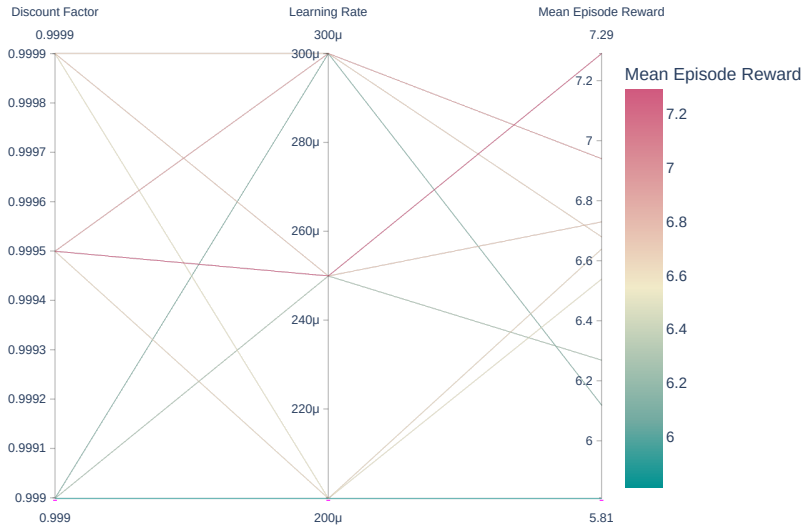


(b) Hyperparameter tuning in stacking task.

Figure 5.11: Hyperparameter tuning results for ALP-GMM



(a) Hyperparameter tuning in reaching task.



(b) Hyperparameter tuning in stacking task.

Figure 5.12: Hyperparameter tuning results for MOC

References

[1] O. G. Selfridge, R. S. Sutton, and A. G. Barto, "Training and tracking in robotics.," in *Ijcai*, 1985, pp. 670–672.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 382, ACM, 2009, pp. 41–48.

[3] K. A. Krueger and P. Dayan, "Flexible shaping: How learning in small steps helps," *Cognition*, vol. 110, no. 3, pp. 380–394, 2009.

[4] J. Schmidhuber, "Curious model-building control systems," in *Proc. international joint conference on neural networks*, 1991, pp. 1458–1463.

[5] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: A short survey," *arXiv preprint arXiv:2003.04664*, 2020.

[6] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," *arXiv preprint arXiv:1707.05300*, 2017.

[7] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *NIPS*, 2016, pp. 1471–1479.

[8] N. Lair, C. Colas, R. Portelas, J.-M. Dussoux, P. F. Dominey, and P.-Y. Oudeyer, "Language grounding through social interactions and curiosity-driven multi-goal learning," *arXiv preprint arXiv:1911.03219*, 2019.

[9] S. Narvekar, J. Sinapov, and P. Stone, "Autonomous task sequencing for customized curriculum design in reinforcement learning.," in *IJCAI*, 2017, pp. 2536–2542.

[10] M. Hessel, J. Modayil, H. van Hasselt, *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI*, AAAI Press, 2018, pp. 3215–3222.

[11] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *arXiv preprint arXiv:2001.06782*, 2020.

[12] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *arXiv preprint arXiv:2003.13661*, 2020.

[13] T. Wang, H. Dong, V. Lesser, and C. Zhang, "Multi-agent reinforcement learning with emergent roles," *arXiv preprint arXiv:2003.08039*, 2020.

[14] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," in *ICLR (Poster)*, OpenReview.net, 2017.

[15] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *arXiv preprint arXiv:1703.05407*, 2017.

[16] C. Blundell, B. Uria, A. Pritzel, *et al.*, "Model-free episodic control," *arXiv preprint arXiv:1606.04460*, 2016.

[17] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[18] T. Galanti and L. Wolf, "On the modularity of hypernetworks," *arXiv preprint arXiv:2002.10006*, 2020.

[19] Y. LeCun, "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27," *OpenReview.net*, 2022.

[20] E. Grefenstette, B. Amos, D. Yarats, *et al.*, "Generalized inner loop meta-learning," *arXiv preprint arXiv:1910.01727*, 2019.

[21] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher–student curriculum learning," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3732–3740, 2019.

[22] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 1312–1320.

[23] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, "Barc: Backward reachability curriculum for robotic reinforcement learning," in *ICRA*, IEEE, 2019, pp. 15–21.

[24] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, Morgan Kaufmann, 1999, pp. 278–287.

[25] A. Goyal, A. R. Didolkar, A. Lamb, *et al.*, "Coordination among neural modules through a shared global workspace," in *ICLR*, OpenReview.net, 2022.

[26] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *arXiv preprint arXiv:1503.08895*, 2015.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[28] O. Ahmed, F. Träuble, A. Goyal, *et al.*, "Causalworld: A robotic manipulation benchmark for causal structure and transfer learning," in *ICLR*, OpenReview.net, 2021.

[29] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *International conference on machine learning*, 2018, pp. 1515–1528.

Chapter 6

Sequence Modeling Enhanced Model Generalization

Chapter 5 provided an in-depth exploration of knowledge distillation techniques and their significance in our models. The advancements discussed here are pivotal in enhancing the efficiency and effectiveness of machine learning systems. Moving forward, Chapter 6 will pivot to a broader examination of model generalization of our proposed models. We will analyze their performance in diverse scenarios, highlighting their versatility and potential in addressing a wide range of real-world challenges.

6.1 Introduction

Recently, with the tremendous success of large language model-based (LLM-based) foundation models [1]–[4], an increasing number of researchers have focused on LLM-based decision-making agents. As shown with GPT-3 [1] and follow-up work [5], [6], the generalization of these LLMs depends significantly on the model size, *i.e.* the number of parameters. This is partly because neural network parameters act as implicit memory [7], enabling models to "memorize" a huge amount of training data by fitting these parameters. However, relying purely on scale has limits, practical and otherwise: there are economic and ecological costs, it reduces accessibility, and more efficient uses of scale might improve performance further. To address some limits of implicit, parameter-based memory in large models, we borrow the concept of "working memory" [8], [9] to explicitly store and recall past experiences for use in future decision-making. The term "working memory" originates from cognitive psychology and neuroscience [8], [10], where it refers to the system responsible for temporary storage and manipulation of information during cognitive tasks.

Our motivation comes from how humans think before they act: they are able to reason on past experience to generate appropriate behavior in new situations. As an illustration, imagine we want to train a robot to play four different Atari

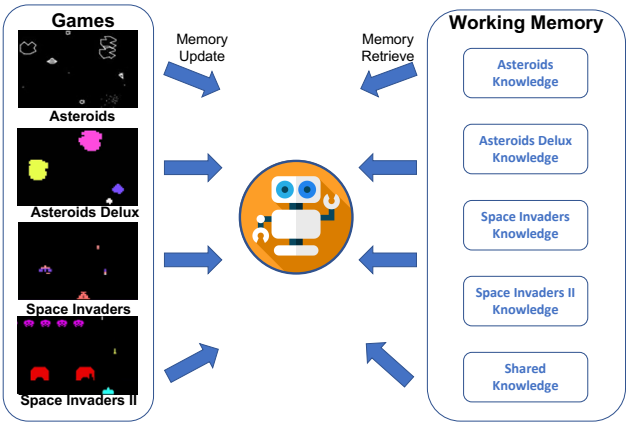


Figure 6.1: A robot uses its working memory to guide its playing strategy.

games: Asteroids, Asteroids Deluxe, Space Invaders, and Space Invaders II (Figure 6.1). Asteroids Deluxe is a sequel to Asteroids that introduces new boss fights and enemies, and the same is true for Space Invaders and Space Invaders II. For the robot to play these four games, it must actively store what it has learned in each game in its working memory and choose the appropriate strategy for each game. Throughout training, the robot's working memory continuously processes and updates relevant game information, allowing it to make informed decisions and adapt its strategies. However, training this robot using implicit memory may cause confusion between similar games and result in incorrect playing strategies. This can ultimately lead to the need for more training time, parameters, and training data.

Thus motivated, we propose **D**ecision **T**ransformers with **M**emory (DT-Mem). We instantiate the internal working memory as a matrix and its functioning entails two primary steps: **memory update** and **memory retrieval**. The memory update involves modifying or replacing existing information. This enables the system to keep track of changes, maintain task-relevant information, and facilitate decision-making. Memory retrieval refers to the process of accessing and recovering stored information. It involves bringing relevant information back to condition decision-making. We use content-based addressing [11]–[13] to locate the memory position to update or retrieve from. To update the memory, we first map the input sequence and memory into three entities: query, key,

and value. Next, we use an attention-based mechanism to calculate the correlations between the input and memory, and then we use the attended weight of the input sequence to update the memory. To retrieve, we read from the updated memory at the content-based address.

Since experience must often be mapped from one task to another (e.g., through analogy in humans) to be useful, we also equip our memory module with an adaptable mapping capability. In particular, we use the low-rank adaptation (LoRA) [14] method in conjunction with a small set of adaptation parameters to modulate the memory module's output. The main idea behind LoRA is to utilize a small amount of labeled data from a new task to learn a low-rank projection matrix. This matrix maps the parameters of a pre-trained model to a new task. We utilize this idea to fine-tune the working memory—via the adaptation parameters—on a new task, using limited data. We fine-tune only the working memory in this work because we rely on the generalization capacity of a pre-trained Decision Transformer (DT). Transformers are often pre-trained on large-scale datasets, as in the case of models like Multi-game DT [15] and Hyper-DT [16], and this pre-training enables them to capture broad knowledge that is transferable across tasks. In contrast, working memory stores task-specific knowledge that should be adapted for new tasks.

DT-Mem differs from external memory and information retrieval-based methods in several ways: (1) memory size, (2) representation of stored information, and (3) retrieval method. In contrast to internal working memory, external memory methods generally

require a large dataset that serves as a look-up table. Each raw data point in the external memory also requires an extra step of representation learning to be input to the neural network. And finally, our working memory relies on an attention-based retrieval method, since attention has demonstrated the ability to generalize across tasks. However, attention is computationally impractical for large sets, and hence external/retrieval-based memory systems tend to rely on k-nearest neighbor (k-NN) search.

DT-Mem builds on earlier work on memory-augmented neural networks [17]—including neural Turing machines [11] and memory networks [18]—in several ways, as we detail in the related work.

To validate our approach, we evaluate DT-Mem on Atari games, as used in Multi-game Decision Transformer (MDT) [15], and Meta-World environments, as used in Prompt Decision Transformer (PDT) [19] and Hyper-Decision Transformer (HDT)[16]. Our results show that DT-Mem improves generalization and adaptability with fewer model parameters and less training time.

We summarize our contributions in the following:

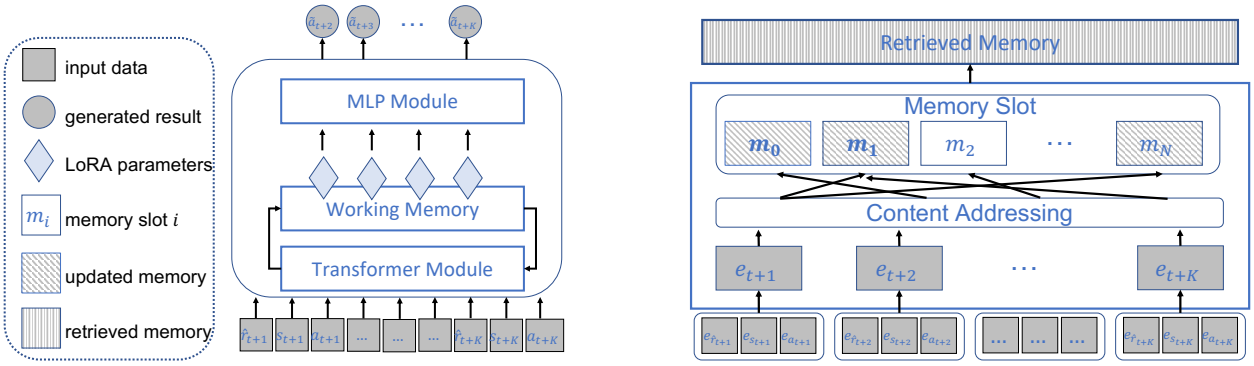
1. We propose **D**ecision **T**ransformers with **M**emory (DT-Mem), a novel Transformer-based DT that improves model generalization, computational efficiency and model efficiency.

2. We introduce a LoRA-based memory module fine-tuning method that further helps DT-Mem adapt to unseen tasks.

In this chapter, we explore various aspects and applications of our proposed models and algorithms. The focus here is not only on demonstrating their effectiveness in specific tasks but also on examining their broader implications and potential applications in diverse scenarios.

- **Initial Sections (6.2 - 6.3):** These sections delve into the detailed analysis and discussion of the primary applications of our models. We evaluate their performance in specific scenarios, providing insights into their strengths and areas for improvement.
- **Further Exploration (Section 6.4):** Moving beyond the initial applications, Section 6.4 presents an exploration into additional domains and scenarios where our models can be applied. This section is crucial in demonstrating the versatility and adaptability of our approaches, extending their applicability to wider contexts.

This chapter aims to present a well-rounded discussion of our models, highlighting not just their core applications but also their potential in extending to various other domains, as elaborated in Section 6.4. This comprehensive approach allows us to fully capture the scope and capabilities of our proposed solutions.



(a) DT-Mem architecture **(b)** Memory module architecture

Figure 6.2: An overview of the proposed DT-Mem architecture. In 6.2a, Transformer module interact with working memory multiple times.

6.2 Methodology

6.2.1 Overview of DT-Mem

In Figure 6.2 we depict the architecture of DT-Mem, which consists of three components: the Transformer module, the Memory module, and the Multi-layer perceptron (MLP) module. The primary role of the Transformer module is to capture dependencies and relationships between states, actions, and returns in a sequence. The input of the Transformer module is a fixed-length sequence of trajectories, denoted as $\tau^{t+1:t+K}$. The output is a sequence of embeddings, where each entry can be attended state embeddings, action embeddings, or return-to-go embeddings. The Transformer module follows the architecture of GPT-2 [20], but without the feed-forward layer after attention blocks. We separate the GPT-2 architecture into two pieces: the Transformer module and the MLP module, following the

setup for natural language processing tasks: one GPT-2 model can be applied to a wide variety of tasks with different MLP modules [20]. Finally, we introduce a working memory module for storing and manipulating intermediate information. This is inspired by the Neural Turing Machine [11], where the memory is utilized to infer multiple algorithms.

6.2.2 Working Memory Module

The design for the working memory is inspired by the way humans think before they act. Its functioning consists of three parts: identifying salient information output from the transformer module, determining where to store new information and how to integrate it with existing memories, and considering how to use these memories for future decision-making. We have broken down these questions and designed the following steps to address them.

Step 0: Working Memory Initialization. The working memory is initialized as a random matrix $\boldsymbol{M}$, where each row $\boldsymbol{m_i} \in \mathbb{R}^d$, with $i \in [0, N]$, represents a memory slot.

Step 1: Input Sequence Organizing. To start, we need to reorganize the input sequence into a different structure. As shown in the problem formulation, the input sequence consists of multiple steps of a tuple $< \hat{r}_t, s_t, a_t >$. Instead of inputting this sequence to the transformer module, we treat each tuple as an entity and embed them in the same space. In other words, we define embedding functions $g_s(s) = e_s$, $g_a(a) = e_a$, and $g_r(\hat{r}) = e_{\hat{r}}$, where $\boldsymbol{e}_s$, $\boldsymbol{e}_a$, and $\boldsymbol{e}_{\hat{r}} \in \mathbb{R}^d$ and d is the dimension in latent space. The final input sequence is the

concatenation of embeddings $\boldsymbol{E} = [\cdots; \boldsymbol{e}_{s_t}, \boldsymbol{e}_{a_t}, \boldsymbol{e}_{\hat{r}_t}; \cdots]$.

Step 2: Content-based Address. We use an attention-based method to locate the correct memory slot for new input by identifying correlated information. This approach is based on the idea that humans tend to store and group similar information together. To locate the memory position, we utilize an attention mechanism. The position address $\boldsymbol{w}$ is calculated as: $\boldsymbol{w} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)$. Here, $\boldsymbol{Q} = \boldsymbol{M}\boldsymbol{W}^q$ and $\boldsymbol{K} = \boldsymbol{E}\boldsymbol{W}^k$, where $\boldsymbol{W}^q$ and $\boldsymbol{W}^k$ are parameters for the Multi-layer perceptron (MLP). The objective is to map the memory and input information into the query and key matrix, and then use the dot product to determine the similarities between these two matrices. The softmax function guarantees that the sum of all addresses equals one.

Step 3: Memory update. To store incoming information and blend it with existing memory, we calculate two vectors: an erasing vector, $\boldsymbol{\epsilon}^e$, and an adding vector, $\boldsymbol{\epsilon}^a$. The erasing vector erases the current memory, while the adding vector controls information flow to the memory. To achieve this goal, we again utilize the attention mechanism. First, we map memory and input information to query, key, and value vectors, denoted as $\hat{\boldsymbol{Q}} = \boldsymbol{M}\hat{\boldsymbol{W}}^q$, $\hat{\boldsymbol{K}} = \boldsymbol{E}\hat{\boldsymbol{W}}^k$, and $\hat{\boldsymbol{V}} = \boldsymbol{E}\hat{\boldsymbol{W}}^v$, respectively, where $\hat{\boldsymbol{W}}^q$, $\hat{\boldsymbol{W}}^k$, and $\hat{\boldsymbol{W}}^v$ are parameters. Next, we calculate the writing strength, $\beta = \text{softmax}\left(\frac{\hat{\boldsymbol{Q}}\hat{\boldsymbol{K}}^T}{\sqrt{d}}\right)$. The erasing vector is used to selectively erase information from the memory matrix and is computed as a function of the content-based addressing vector and the write strength. The erasing vector is calculated as $\boldsymbol{\epsilon}^e = \boldsymbol{w}(1-\beta)$. The complement of the write strength is 1 minus the write strength, so this

will result in a vector where the elements corresponding to the selected memory locations are set to 0, and the elements corresponding to the unselected memory locations are unchanged.

The adding vector is used to selectively add information to the memory matrix and is computed as a function of the write strength and the input vector. Specifically, the adding vector is calculated as $\boldsymbol{\epsilon}^a = \boldsymbol{w}\beta\hat{\boldsymbol{W}}^v x$.

Finally, the memory is updated as $\boldsymbol{M}_t = \boldsymbol{M}_{t-1}(\boldsymbol{I} - \boldsymbol{\epsilon}^e) + \boldsymbol{\epsilon}^a$. If the selected memory slot is empty or erased, the new information will be stored. Otherwise, the new information will be blended with the existing memory contents.

Step 4: Memory retrieve To utilize memory for decision-making, we retrieve information from the updated memory slot. Reading from the memory matrix is done by computing a read position vector. This vector can be computed using the above content-based addressing mechanism that involves comparing the query vector with the contents of the memory matrix. Note that in other retrieval-based methods [21], [22], nearest neighbor is the common way to retrieve related information. However, in our case, the internal working memory is smaller than the typical external working memory, which makes attention-based retrieval feasible. Since the query information is the same as the input information, we use the same content address to retrieve the memory: $\boldsymbol{E}_{out} = \boldsymbol{w}\boldsymbol{M}_t$.

6.2.3 Pre-training DT-Mem

We use a set of training tasks T^{train}, where each task $\mathcal{T}_i \in T^{train}$ has an associated offline dataset $\mathcal{D}_i$ consisting of hundreds of trajectories τ generated by a behavior policy. The behavior policy can be either a pre-trained policy (such as DQN) or a rule-based policy, depending on what is available. Each trajectory $\tau = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$, where $s_i \in \mathcal{S}, a_i \in \mathcal{A}, r_i \in \mathcal{R}$, and H is the episode length.

To serve as an input to the DT-Mem, we first segment the trajectory τ into several pieces, each with length K. We denote $\tau_{t+1:t+K} = (s_{t+1}, a_{t+1}, r_{t+1}, \cdots, s_{t+K}, a_{t+K}, r_{t+K})$ as one of the input sequence. However, we modify these trajectories instead of inputting them directly. Specifically, we follow the return-to-go Decision Transformer idea [23] and calculate the return to go, $\hat{r}_t = \sum_{t=t+K}^{t+1} r_t$, for every timestep. This is effective because $\hat{r}_t$ acts as a subgoal. It encourages the Transformer module to generate actions that can reduce this value as close to zero as possible. Then we input the modified trajectories $\hat{\tau}_{t+1:t+K} = (\hat{r}_{t+1}, s_{t+1}, a_{t+1}, \cdots, \hat{r}_{t+K}, s_{t+K}, a_{t+K})$ to the transformer module. The output of the transformer module is a sequence embedding $e_{seq} \in \mathbb{R}^{d \times 3K}$, where d is the dimension of the embedding space.

Next, we transmit e_{seq} to the Working Memory module to update and retrieve the memory information. Finally, we use the retrieve memory $\boldsymbol{E}_{out}$ and MLP modules to generate the corresponding actions $\hat{a}_t$. We minimize a supervised training loss with three terms: predicted

actions $\tilde{a}_t$, predicted reward $\tilde{r}_t$, and predicted return-to-go $\tilde{R}_t$. The loss function is:

$$\mathcal{L} = \sum_{t+1}^{t+K} ||\tilde{a}_t - a_t||^2 + \alpha||\tilde{r}_t - \hat{r}_t||^2 + \lambda||\tilde{R}_t - r_t||^2, \tag{6.1}$$

where α and λ are scalar hyper-parameters. In experiments, we found that the final performance is not sensitive to these two hyper-parameters, so we set them to 1 for simplicity.

6.2.4 Fine-tuning DT-Mem with LoRA

Fine-tuning LLMs involves heavy computation due to the large number of parameter updates required. We argue that fine-tuning only the working memory can achieve results comparable to those of fine-tuning the entire parameter space. LLMs benefit from being trained on large-scale datasets, which expose the model to a diverse range of linguistic patterns and semantic relationships, such as models like [24] or GPT [20]. This exposure helps the model learn robust and generalized representations that can capture different aspects of language understanding and generation. After pre-training, the model can be fine-tuned on specific downstream tasks with task-specific labeled data. In our case, this task-specific knowledge is stored in working memory. Thus, fine-tuning the working memory helps the model update its working memory to adapt to the new task.

We apply the low-rank adaptation approach [14] to fine-tune the working memory module.

Specifically, we modify the forward pass by adding low-rank matrices to $\boldsymbol{W}^q$, $\boldsymbol{W}^k$, $\boldsymbol{W}^v$, $\hat{\boldsymbol{W}}^q$, and $\hat{\boldsymbol{W}}^k$. Let's take $\boldsymbol{W}^q$ as an example. Assuming the original output for query information is $\boldsymbol{Q} = \boldsymbol{M}\boldsymbol{W}^q$, we adapt this query value to a new task as $\boldsymbol{Q'} = \boldsymbol{M}(\boldsymbol{W}^q + \boldsymbol{B}^q\boldsymbol{A}^q)$, where $\boldsymbol{W}^q \in \mathbb{R}^{n\times d}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$, and $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, and m is the size of the working memory. Since the rank $m \ll min(n, d)$, fine-tuning the parameters $\boldsymbol{B}^q$ and $\boldsymbol{A}^q$ reduces the number of trainable parameters for downstream tasks. We perform supervised training by computing the loss between the model's output and the labels in the fine-tuning dataset. During this process, only $\boldsymbol{B}^q$ and $\boldsymbol{A}^q$ are updated.

6.3 Evaluation on Games

We designed our experiments to answer the following questions:

- **Q1:** Does DT-Mem improve model generalization?
- **Q2:** Does DT-Mem improve networking and training efficiency?
- **Q3:** Does fine-tuning only the memory module improve model adaptability?

Recall that we use generalization to refer to performance on tasks the model has never trained on (zero-shot), and adaptability to refer to performance after fine-tuning.

6.3.1 Environments and Models Setup

Atari Games To ensure a fair comparison with the Multi-Game Decision Transformer, we used the same Atari dataset[1], which comprises multiple training runs of DQN trajectories. Due to limited compute resources and to prevent cherry picking, we selected 17 games from the available 41 based on their alphabetical order, as introduced in [15]. For each game, the data contains 50 policy checkpoints, each of which contains 500k environment steps. For the fine-tuning dataset, we randomly selected 10% of the data from the unseen dataset, which yielded 50k environment steps. Following the settings from [15], we choose five games (Alien, Ms. Pac-Man, Pong, Space Invaders and Star Gunner) to be used only for fine-tuning. Moreover, [25] suggests that return-conditioned supervised learning (RCSL) algorithms require strong dataset coverage to select a near-optimal policy. Therefore, our dataset contains both expert and non-expert behaviors.

Meta-World To make a fair comparison with Hyper-DT and Prompt-DT, we evaluate the proposed method on the Meta-World environment [26]. We conducted the evaluation using the Meta-World ML45 benchmark, which includes 45 training tasks and 5 testing tasks. Following the approach taken in [16], for each training task, we generated an offline dataset containing 1000 episodes for each game, using a rule-based script policy. For fine-tuning data, we randomly pick 10k episodes from the testing dataset, as compared to 20k-80k episodes used in Hyper-DT.

[1] https://research.google/tools/datasets/dqn-replay/

DT-Mem settings We report results for DT-Mem 20M (20 million parameters), which consists of 13M transformer parameters and 7M memory module parameters.

Training and Fine-tuning For all games, we use eight V100 GPUs for model training and one V100 GPU for fine-tuning. We train on both Atari games and Meta-World for 10M steps. For fine-tuning on unseen scenarios, we train for 100k steps.

6.3.2 Baseline Methods

We compare DT-Mem's performance against the following baselines.

MDT Multi-game Decision Transformer [15], which trains a large transformer-based model on multi-game domains.

HDT Hyper-Decision Transformer [16], which utilizes a hyper-network module to help DT adapt rapidly to unseen tasks. Since we do not have access to the implementation at the time of writing, for the sake of correctness, we compare our model with HDT on Meta-World only. The results reported in our evaluation section come from the HDT paper.

PDT The Prompt Decision Transformer [19] generates actions by considering both recent context and pre-collected demonstrations from the target task.

6.3.3 DT-Mem improves model generalization.

We evaluate five held-out games fine-tuning results as listed in Table 6.1. Each evaluation signifies an average derived from 16 runs, each under differing random seeds. The derived

	Alien	MsPacman	Pong	SpaceInvaders	StarGunner
MDT	3.8% (±0.4%)	13.2% (±1.3%)	0% (±0%)	8.6% (±1.6%)	2.3% (±0.1%)
RMDT	22.3% (±10.7%)	22.9% (±8.9%)	0% (±0%)	17.6% (±9.2%)	27.7% (±11.5%)
DT-Mem	51.0% (±32.2%)	69.3% (±19.3%)	0% (±0%)	53.6% (±29.0%)	62.2% (±19.1%)

Table 6.1: Evaluation results on 5 held-out games after pre-training on other Atari Games. Each value represents the DQN-normalized score, computed with a 95% confidence interval.

results show that the memory-incorporated method, RMDT and DT-Mem, enhances model generalization when compared to their ablation method MDT. A noteworthy observation is that DT-Memdemonstrates superior generalization performance than RMDT in four out of the five games. Neither of the methods achieves a good result in "Pong". We further discuss whether fine-tuning helps to improve the performance in Section 6.3.5.

6.3.4 DT-Mem enables more computationally efficient training.

To demonstrate training efficiency, we illustrate the model training time in Table 6.2 During training, we found that DT-Mem reduces the training time by approximately 4 times, 8 times, and 32 times compared to MDT-13M, MDT-40M, and MDT-200M, respectively. For the training curve, it is reasonable to report the prediction loss on the

Model	Training time (hours)
DT-Mem	**50**
MDT-13M	200
MDT-40M	400
MDT-200M	1600

Table 6.2: Model training time

training dataset since we use a supervised loss.

Here, the prediction accuracy consists of three parts: action prediction accuracy, reward prediction accuracy and return prediction accuracy.

6.3.5 Fine-tuning only the memory module improves model adaptability.

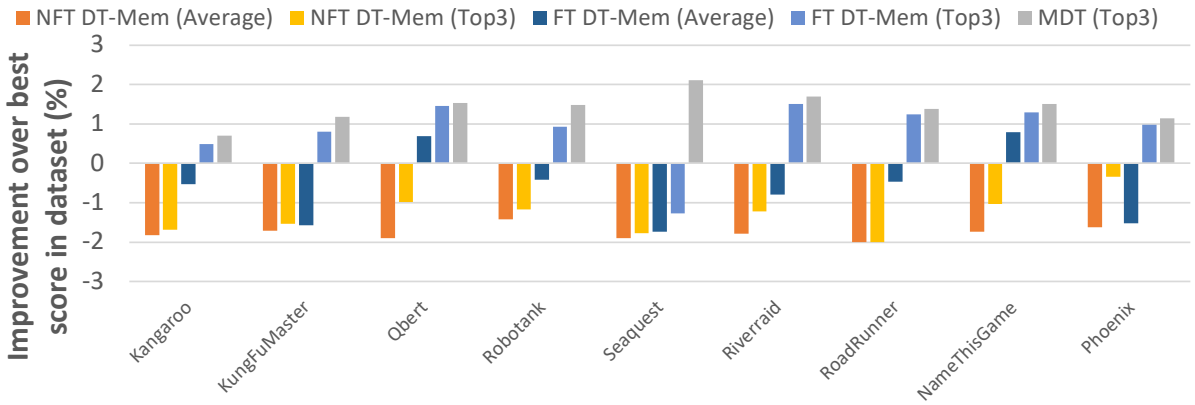


Figure 6.3: Fine-tuning performance on 10% of dataset in unseen Atari games. NFT stands for no fine-tune model and FT stands for fine-tune model. Note that these games *are* in the training dataset of MDT. The y-axis is the logarithm of the improvement percentage.

Another question we care about is how the pre-trained DT-Mem performs on unseen tasks. We randomly selected nine unseen Atari games and evaluated their performance through relative improvement scores, as shown in Figure 6.3. Without fine-tuning, DT-Mem cannot compete with the human-best scores across the dataset. After fine-tuning with 10% of the unseen data, DT-Mem-Top3 surpasses the human-best scores in eight out of nine games, while DT-Mem-Average only outperforms the human-best scores in two out of nine

games. It is reasonable that none of the proposed methods can compete with MDT-Top3, since MDT was trained on these nine games with the full dataset. Thus, DT-Mem with simple fine-tuning yields promising performance, demonstrating its generalization and adaptability.

To compare the generalization of MDT and DT-Mem, we evaluated 5 held-out games that were not included in either model's training dataset. We observed that no-fine-tune DT-Mem failed to achieve good results in all 5 games for both average and top 3 rollouts. After fine-tuning, the average rollout results of DT-Mem outperformed the DQN score in 3 out of 5 games and achieved similar performance compared to MDT in Alien, Ms. Pac-Man, and StarGunner games. The top3 DT-Mem rollouts results outperformed MDT-Top3 in 4 out 5 games and increase the DQN-normalized score on average by 15.5%. This result is an indication of the effectiveness of the proposed method. However, we also noticed that fine-tuning DT-Mem on the Pong game did not produce good results. We hypothesize that the limited number of training games is the reason. To mitigate this issue, we increased the fine-tuning datasets from 10% to 20% and fine-tuning steps from 100k steps to 200k steps. After fine-tuning on more data and steps on Pong, results show that when compared to MDT-Top3, using DT-Mem on average decreased performance by 0.978%, but using DT-Mem Top3 increased performance by 1.154%. In conclusion, our findings suggest that the proposed DT-Mem improves the generalization of the model, especially in games that are not included in the training dataset. However, the effectiveness of fine-tuning may depend on the number of training games and the amount of fine-tuning steps. Therefore, future research

should explore the optimal combination of these factors to further enhance the performance of the model.

To further understand the adaptability of the proposed method, we compare DT-Mem with HDT and PDT in meta-world environments. The quantitative fine-tuning results are shown in Table 6.3. Overall, DT-Mem achieves the best performance in the comparison. As we can see, compared to HDT, DT-Mem increases both training, testing (no-FT) and testing (FT) scores by an average of 3%, 8% and 3%, respectively. Moreover, the HDT adaptation module (hyper-network module), while small (69K) relative to the full model (13M), relies on the pre-trained hyper-network, which contains 2.3M parameters. We argue that the hyper-net is more burdensome than our design: it uses more than 10x the number of adaptation parameters (147K) used by DT-Mem and requires an extra compute phase to pre-train the hyper-network module.

	Model Sizes		Meta-World ML45 Performances		
	Adaptation	Percentage	Train	Test (no-FT)	Test (FT)
HDT	69K	0.5%	0.89 ± 0.00	0.12 ± 0.01	0.92 ± 0.10
PDT	6K	0.05%	0.88 ± 0.00	0.06 ± 0.05	0.09 ± 0.01
DT-Mem	147K	0.7%	$\mathbf{0.92 \pm 0.00}$	$\mathbf{0.20 \pm 0.01}$	$\mathbf{0.95 \pm 0.10}$

Table 6.3: Evaluation results on Meta-World ML45 benchmarks

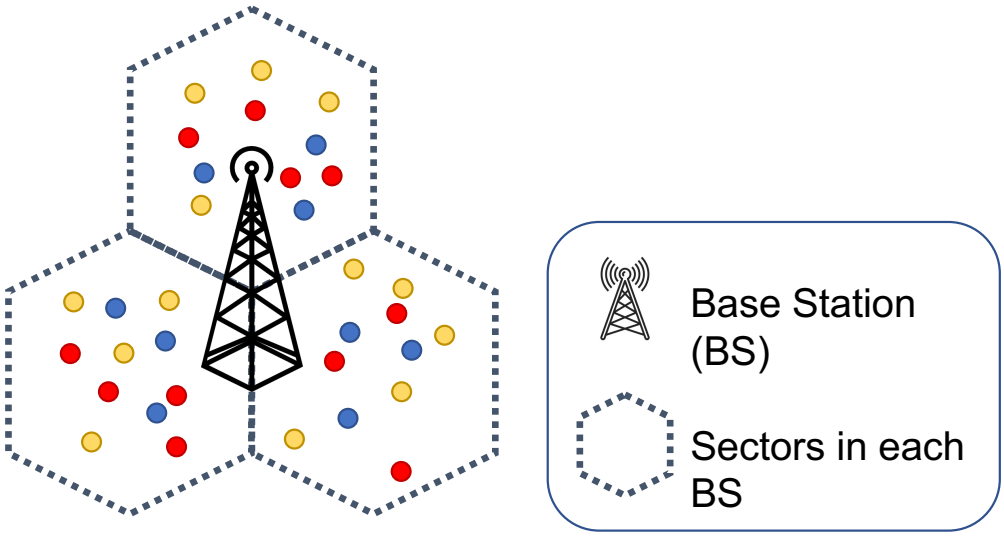


Figure 6.4: The simulation scenario contains one BS. Each hexagon represents one cell, which controls 120 degrees. The yellow, red, green, and blue dots stands for idle UEs, active-downlink UEs, active-handover UEs, and inactive UEs, respectively.

6.4 Evaluation on Network Load Balancing

6.4.1 Environment Setup

To evaluate the effectiveness of our approach for load balancing, we conducted experiments in a communication network simulator. The simulator is designed to emulate a real-world network environment, and it includes various traffic scenarios with different numbers of UEs and packet sizes. As shown in Fig. 6.4. The load balancing task was performed on a network consisting of several Base Stations (BS), each of which consists of 3 sectors, each with 4 cells operating on different frequency channels. In total, we evaluated 26 traffic scenarios in the simulator, each representing different network conditions. The scenarios were chosen

to represent a diverse range of network conditions, including low and high network traffic, as well as different types of packet sizes. This allowed us to test the effectiveness of our approach across a range of different network conditions.

6.4.2 System Performance Metrics

In this paper, we evaluate the system performance using several metrics over a period of interest denoted by T. We define $u_{i,k}$ as the k-th UE in the i-th cell, and $A_{i,k}$ as the total size of packets received by UE $u_{i,k}$.

- **Minimum Throughput (minTput)** $G_{minTput} = \min_{i,k}\left(\frac{A_{i,k}}{T}\right)$ shows the worst-case UE performance. Maximizing this metric improves the worst-case user experience.
- **Total Throughput (totalTput)** $G_{totalTput} = \sum_i \sum_k \frac{A_{i,k}}{T}$ evaluates overall system performance. This metric reflects the overall provided network services.

6.4.3 Methods Evaluated

We compare our method with the following SOTA methods.

- **Rule-based** manages the base station load balancing using pre-programmed control parameters according to prior knowledge [27].
- **PPO** (proximal policy optimization) [28] is one of the SOTA policy update algorithms[2]

[2]Compared with other SOTA RL algorithms such as TD3 [29] and SAC [30], we find PPO achieves the best system performance overall metrics, which we choose as the compared schemes here.

that has been widely used in control problems. To make a fair comparison, we train PPO on several representative traffic scenarios and evaluate them on other unseen scenarios.

- **BC** (Behavior Cloning) [31] is a method that involves training a policy to imitate or replicate an existing behavior policy. The aim of behavior cloning is to produce a model that can replicate the expert's behavior on new, unseen data, with the hope of achieving similar or better performance than the expert.

- **CQL** (Conservative Q-Learning) [32] is another important SOTA offline RL approach, which learns a Q-function that is conservative with respect to the data distribution. For fair comparison, we substitue the Q-learning algorithm with PPO method. CQL serves as our primary comparison.

- **HDT** (Ours): as point out in [19] we use 20% of the trajectories as the task information. In our case, the total length of the trajectories is 24, which means the task information contains the first 5 timesteps.

6.4.4 Training Datasets Preparation

Our simulator consists of a total of 100 traffic scenarios, each with different parameter settings such as number of UEs, packet size, and request interval. Due to page limits, we are unable to include all the details of the traffic settings in the paper. To provide a basic

	Number of UEs	Packet Size	Request Interval
#21	35	1.2	180
#22	40	0.2	40
#23	40	0.1	9
#24	35	0.4	40
#25	40	2	280
#26	35	1	120
#27	55	0.8	80
#28	30	1.2	60

Table 6.4: The environment parameters of 8 traffic patterns.

understanding, we summarize 8 out of the 100 traffic scenarios in TABLE 6.4.

To prepare the datasets, we follow the data-collection framework proposed in [33]. The framework stores the training trajectories (s_t, a_t, r_t) for every timestep t. In this paper, we use the PPO method as the training policy, as it has shown the best performance among all state-of-the-art model-free RL algorithms. Following the data splitting convention, we randomly separate the 100 training scenarios into 80 training scenarios and 20 testing scenarios. This results in 80% and 20% for training and testing, respectively.

6.4.5 Evaluation Results on Metrics

We first show the collected mean episode rewards during the model training phase. As shown in Fig. 6.5, the proposed method HDT achieves the highest rewards as the training going on. CQL method achives around the same mean episode rewards as collected in the datasets. BC method cannot achieve the best reward stored in the datasets.

We compared the zero-shot adaptation abilities of different schemes, and due to the

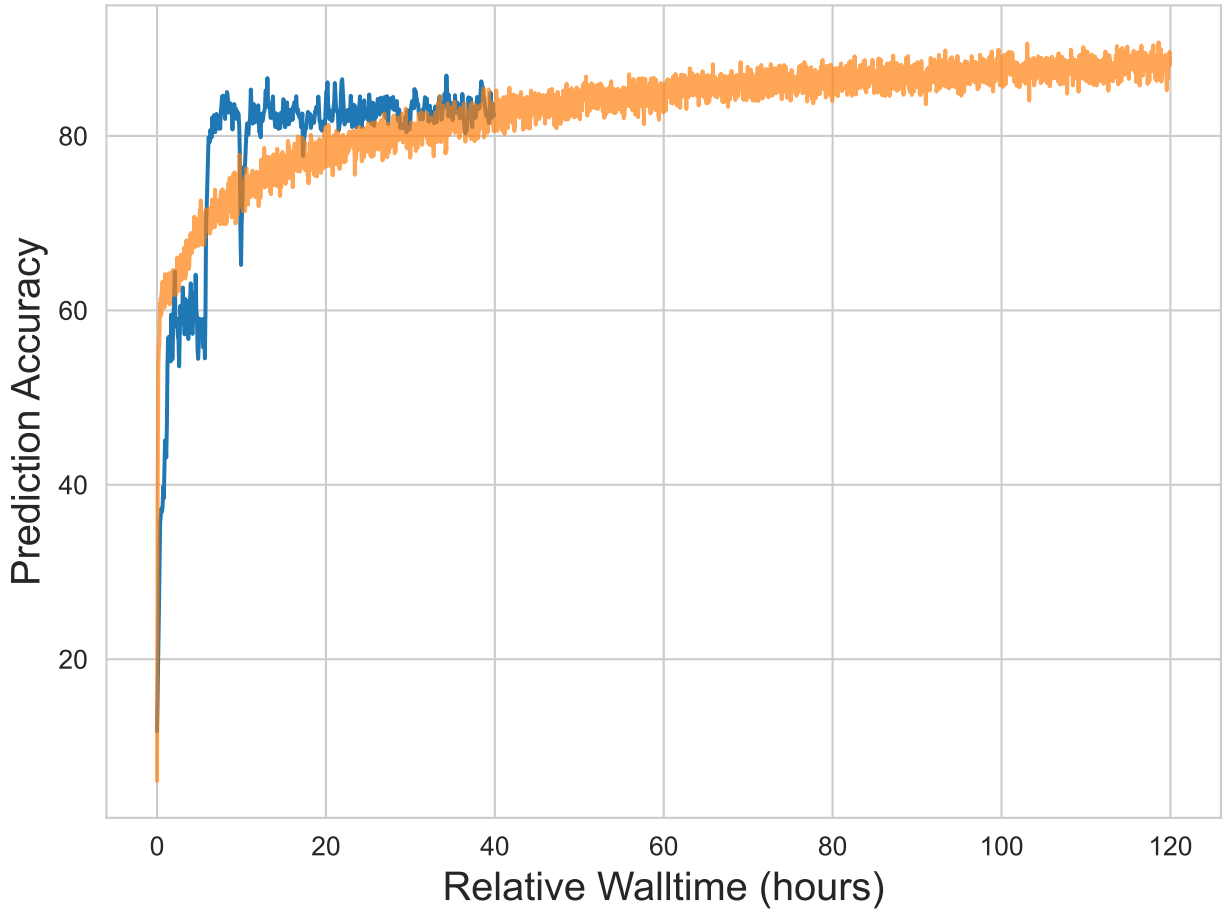


Figure 6.5: Collected mean episode rewards during training. Each curve represents a offline-RL method. The purple dash line denotes the best rewards stored in the datasets, which is collected by PPO method.

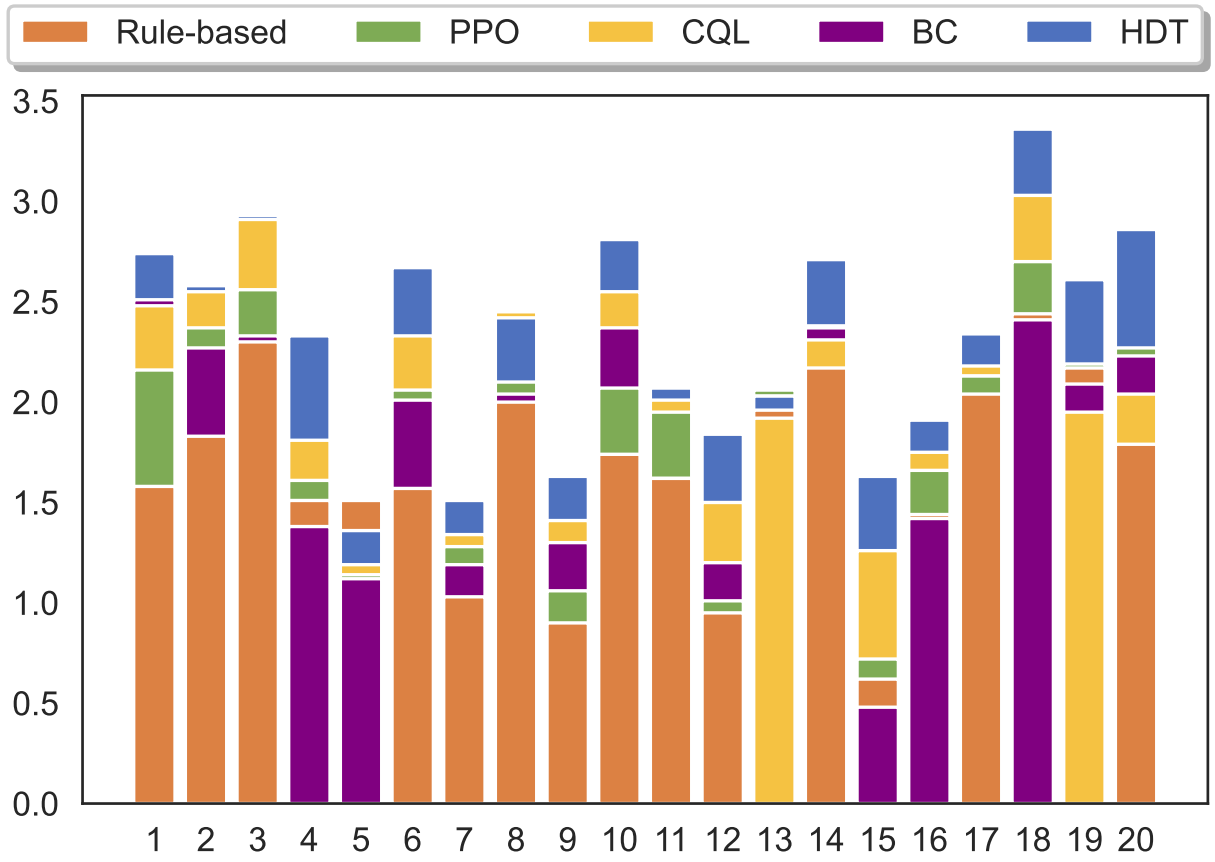


Figure 6.6: Comparison of minTput metric results for 20 unseen traffic scenarios. For better observation, we stack different methods in one bar plot with overlapping. The higher results show better performance.

page limit, we only report one performance metric in this result. However, the phenomenon remains consistent across all metrics [3]. As shown in Fig. 6.6, HDT achieves the best minTput values in 17 out of 20 scenarios, demonstrating its zero-shot generalizability. In scenario 5, none of the baselines achieve competitive results compared to the Rule-based method. We performed further analysis on this scenario and found that it is different from other traffic patterns. In scenario 8, CQL receives mean episode rewards of 2.45, which is slightly better than HDT's 2.42. Since CQL is the state-of-the-art offline RL algorithm, this result is reasonable and does not affect the main observation. The same conclusion applies to scenario 13, where the PPO method achieves the best reward of 2.06, slightly better than HDT's 2.03. The reason for this is that the PPO method is overfitted to the training scenarios and can achieve good results only if the testing scenarios are similar to the training datasets.

We further evaluated the effectiveness of the proposed component in the ablation study. To illustrate the effectiveness of the proposed method, we added a variation of HDT without the hypernet module and named it vanilla-HDT. As shown in Table 6.5, both DT and HDT outperform CQL and improve performance by an average of 8.9% and 15.1%, respectively. The reason can be summarized in two parts: (1) the transformer module takes advantage of the whole existing trajectories to make the current decision. Unlike classical RL methods that choose actions based on current state, DT utilizes the trajectories $\tau_{0:t-1}$ up to the current timestep t and makes decisions accordingly, and (2) the hypernet module further improves

[3]This also applies to the results shown in the ablation study

the adaptability of DT by utilizing the task information and inferring the parameter settings for the MLP module.

Traffic ID	Rule-based	CQL	vanilla-HDT	HDT
1	3.09	4.07	4.42	4.85
2	3.43	4.14	4.23	4.66
3	4.07	4.24	4.33	4.61
4	2.78	3.03	3.35	3.51
5	1.49	1.64	2.4	2.52
6	2.48	3.02	2.86	3.05
7	2.49	2.65	2.66	2.95
8	3.13	3.25	3.61	3.62
9	2.41	2.39	2.41	2.63
10	2.89	3.3	3.87	4.02
11	3.03	3.21	3.48	3.55
12	2.49	2.69	3.71	3.95
13	3.7	4.07	4.42	4.57
14	3.62	4.14	4.77	4.92
15	1.73	1.81	2.1	2.61
16	2.51	2.61	3.21	3.4
17	3.92	4.06	4	4.17
18	3.78	3.98	4.05	4.28
19	3.94	5.21	5.58	5.63
20	3.28	3.38	3.36	3.51
Average	3.01	3.34(+11%)	**3.64**(+21%)	**3.85**(+61%)

Table 6.5: Ablation study on totalTput metric results for 20 unseen traffic scenarios. Each value shows the mean episode reward during evaluation. The relative average improvements over baselines are shown in brackets.

6.5 Implementation Details

6.5.1 DT-Mem network architecture

Table 6.6 summarizes the different model configurations used for evaluation. In this section, we describe these model configurations in detail. While Table 6.6 provides a summary, we will also provide additional information here. DT-Mem, PDT and HDT are all share the same transformer architectures. However, for task-adaptation, HDT utilizes a pre-trained 2.3M hyper-network, while DT-Mem introduces 147K LoRA parameters. To compare with MDT, we use the same parameter size as reported in [15].

Model	Layers	Hidden size (d)	Heads	Params	Memory Size
HDT	4	512	8	13M	N.A.
MDT-200M	10	1280	20	200M	N.A.
DT-Mem	4	512	8	13M	559K

Table 6.6: Detailed Model Sizes

6.5.2 Hyper-parameters

In this section, we will delve into the specifics of the model parameters. Understanding these parameters is key to understanding the workings of the model. It is worth noting that the source code for this model is publicly available at `https://github.com/luciferkonn/DT_Mem/tree/main`. This allows for a deeper understanding of the model's inner workings and may facilitate the replication of its results.

Hyperparameters	Value
K (length of context)	28
dropout rate	0.1
maximum epochs	1000
steps for each epoch	1000
optimizer learning rate	1e-4
weight decay	1e-4
gradient norm clip	1.
data points for each dataset	500,000
batch size	64
memory slots	1290
activation	GELU
optimizer	AdamW
scheduler	LambdaLR

Table 6.7: Hyperparameters for DT-Mem training

6.5.3 Training and fine-tuning algorithm

In this section, we present the pre-training DT-Memin Appendix 6.5.3 and fine-tuning DT-Mem with LoRA in Appendix 6.3.5.

We pre-train DT-Mem on multiple offline datasets. Each gradient update of the DT-Memmodel considers information from each training task.

We fine-tune the memory module to adapt to each downstream task. To achieve this, we fix the pre-trained DT-Mem model parameters and add additional LoRA parameters for the memory module feed-forward neural networks. The fine-tune dataset is used to update these LoRA parameters only.

Algorithm 3: Pre-train DT-Mem

for *T episodes* **do**
  for *Task* $\mathcal{T}_i \in T^{train}$ **do**
    Sample trajectories $\tau = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$ from the dataset $\mathcal{D}_i$.;
    Split trajectories into different segments with length K and calculate return-to-go in the input sequence.;
    Given $\hat{\tau}_{t+1:t+K}$, compute the sequence embedding e_{seq}.;
    Update the working memory and retrieve the relative information as $\boldsymbol{E}_{out}$;
    Given $\boldsymbol{E}_{out}$, predict actions $\tilde{a}_t$, reward $\tilde{r}_t$, and return-to-go $\tilde{R}_t$.;
    Compute the loss according to Eqn. 6.1.;
    Update all modules parameters.;

Algorithm 4: Fine-tuning DT-Mem

Require: Fine-tuning dataset $\mathcal{T}^i \in T^{test}$ dataset $\mathcal{D}^i$ for $\mathcal{T}^i$. Initialize LoRA parameters $\hat{\boldsymbol{B}}^q, \hat{\boldsymbol{B}}^k, \hat{\boldsymbol{B}}^v, \hat{\boldsymbol{A}}^q, \hat{\boldsymbol{A}}^k, \hat{\boldsymbol{A}}^v, \boldsymbol{B}^q, \boldsymbol{A}^q, \boldsymbol{B}^k, \boldsymbol{A}^k$. **for** *T steps* **do**
  Split trajectories into different segments with length K and calculate return-to-go in the input sequence.;
  Given $\hat{\tau}_{t+1:t+K}$, compute the sequence embedding e_{seq}.;
  Update working memory using $\hat{\boldsymbol{Q}} = \boldsymbol{M}(\hat{\boldsymbol{W}}^q + \hat{\boldsymbol{B}}^q\hat{\boldsymbol{A}}^q)$,
  $\hat{\boldsymbol{K}} = \boldsymbol{M}(\hat{\boldsymbol{W}}^k + \hat{\boldsymbol{B}}^k\hat{\boldsymbol{A}}^k)$,$\hat{\boldsymbol{V}} = \boldsymbol{M}(\hat{\boldsymbol{W}}^v + \hat{\boldsymbol{B}}^v\hat{\boldsymbol{A}}^v)$,
  $\boldsymbol{Q} = \boldsymbol{M}(\boldsymbol{W}^q + \boldsymbol{B}^q\boldsymbol{A}^q)$,$\boldsymbol{K} = \boldsymbol{M}(\boldsymbol{W}^k + \boldsymbol{B}^k\boldsymbol{A}^k)$;
  Retrieve the relative information as $\boldsymbol{E}_{out}$;
  Given $\boldsymbol{E}_{out}$, predict actions $\tilde{a}_t$, reward $\tilde{r}_t$, and return-to-go $\tilde{R}_t$.;
  Compute the loss according to Eqn. 6.1.;
  Update LoRA parameters only.;

6.5.4 Evaluation Parameters

To evaluate the performance of our model on Atari games, we randomly selected 16 different random seeds for evaluation. We chose the random seed by multiplying the number of runs by 100. For example, the random seed for run 6 is $6 \times 100 = 600$.

6.5.5 DT-Mem improves training performance.

We want to evaluate pre-training whether adding the working memory module helps improve the pre-training performance. Thus, we choose relative improvement: $\text{rel-imp}(\%) = (\text{score}_{model} - \text{score}_{dataset})/\text{score}_{dataset} \times 100$ to measure the model performance. As shown in Figure 6.7, the proposed DT-Mem-Top3 out performs MDT-Top3 in 13 out of 17 games. DT-Mem-Average outperforms MDT-Top3 in 6 out of 17 games. These results demonstrates the effectiveness of the proposed method.

6.5.6 Training Efficiencies

To demonstrate training efficiency, we illustrate the model training curve in Figure 6.8. For the training curve, it is reasonable to report the prediction loss on the training dataset since we use a supervised loss. Here, the prediction accuracy consists of three parts: action prediction accuracy, reward prediction accuracy and return prediction accuracy. The y-axis shows the average value of these three predictions, and the x-axis is the relative walltime based on same computing resources.

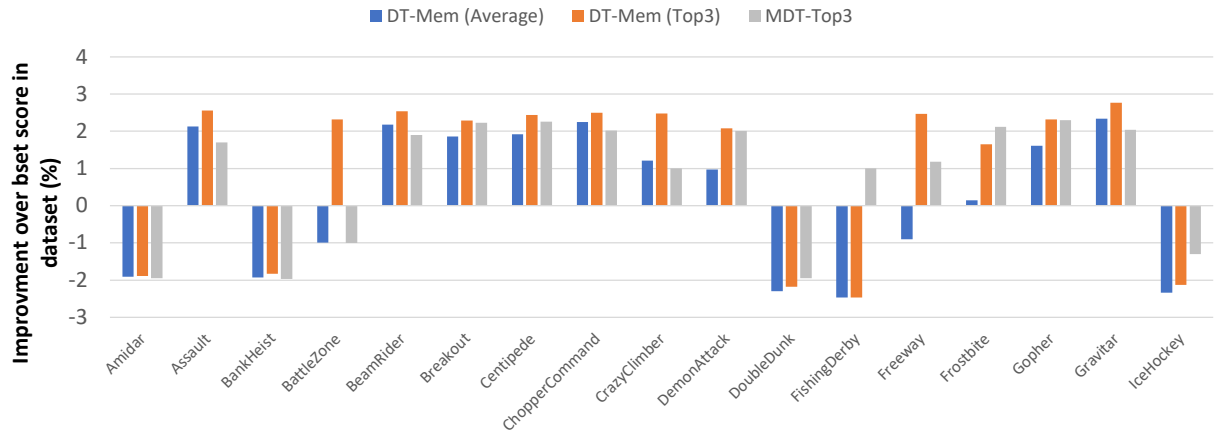


Figure 6.7: The percent improvement for training dataset. We take the logarithm of the original improvements for better visualization. The evaluation are done in 16 runs with different random seeds. Average stands for the mean value of 16 runs. Top3 represents the top 3 rollouts out of 16 runs.

6.5.7 The analysis of memory size

In this section, we investigate the impact of the memory module size on the performance of DT-Mem. We employ the Bayes optimization strategy to tune the parameters. It's worth noting that the memory size is calculated by multiplying the number of memory slots by the size of each slot, which is fixed at 512 dimensions for the sake of evaluation simplicity. To expedite the hyper-parameter tuning process, we present the evaluation results based on 100k training steps of the StarGunner game. We assess various configurations of memory slots and calculate their corresponding average rewards over 16 runs. Figure 6.9 reveals several key findings: (1)Increasing the size of memory slots leads to a higher reward accumulation. Notably, there is a significant performance boost when the number exceeds 1200. (2)In

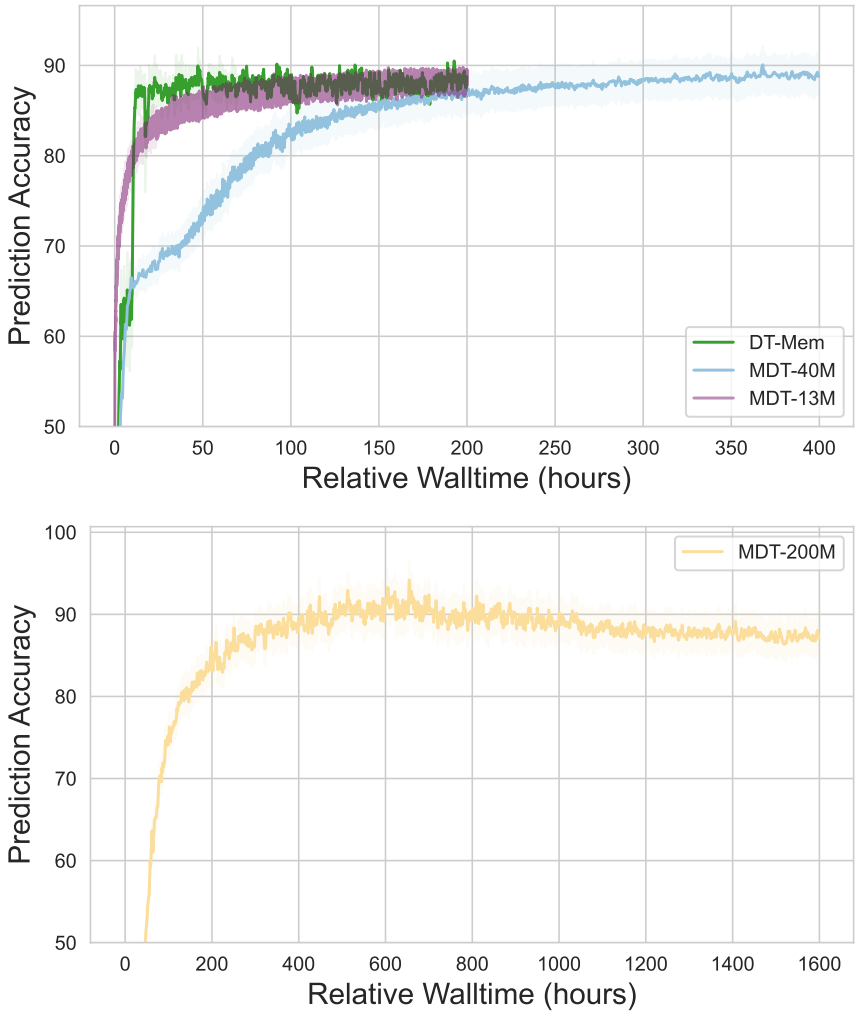


Figure 6.8: This graph shows the prediction accuracy during training. Each curve represents three runs with different random seeds. For better visualization, MDT-200M is displayed in a separate figure.

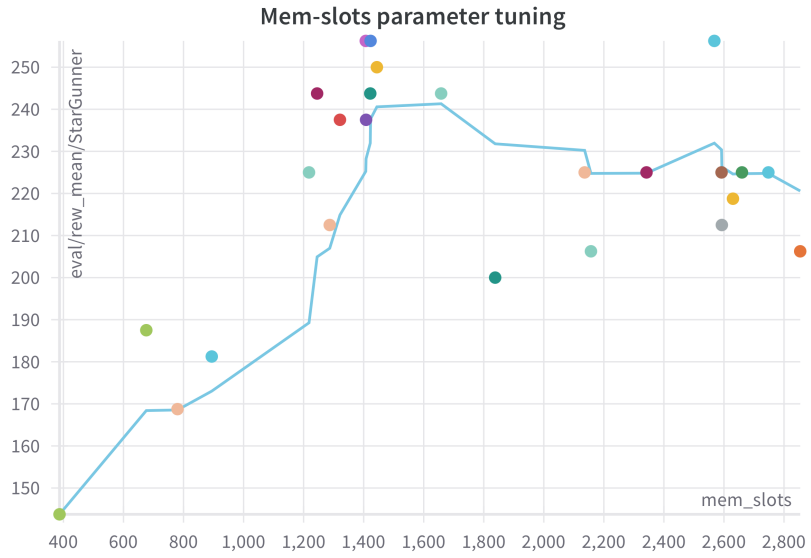


Figure 6.9: The parameter tuning results for the number of memory slots. The blue curve shows the like from left to right over the x axis and plots the running average y value.

summary, when the number of memory slots exceeds 1800, the performance of the system decreases. This decline occurs because there is a trade-off between the number of memory slots and the training steps. With a larger number of memory slots, it becomes necessary to allocate more training time.

6.5.8 Ablation study of LoRA adaptor

In this section, we conduct an ablation study of LoRA-based memory adaptor. We substitute LoRA adaptor with hyper-networks. Specifically, the parameters of the memory module are generated from hyper-networks. This approach is based on [34], where hyper-networks take task-related information as input and generate the corresponding

	Meta-World ML45 Performances			Data size	Model	
	Train	Test (no-FT)	Test (FT)		Adap.	Per.
DT-Mem (hyper-net)	0.92 ± 0.01	$\mathbf{0.23 \pm 0.10}$	0.81 ± 0.15	30	5.7M	43.8%
DT-Mem	$\mathbf{0.92 \pm 0.00}$	0.20 ± 0.01	$\mathbf{0.95 \pm 0.10}$	10	147K	0.7%

Table 6.8: Ablation study results on Meta-World ML45 benchmarks. DT-Mem (hyper-net) denotes the variation of DT-Mem, which substitute LoRA adaptation module with hyper-networks. Adap. stands for adaptation parameters, and Per. stands for percentage of original model.

networks for the downstream MLP. We use the same approach and generate parameters that are conditioned on two types of inputs: the task embedding from the task encoder and the sequence embeddings from the Transformer module.

To generate task embeddings, we adopt the same idea from PDT [19], which demonstrates that a small part of trajectories can represent the task-related information. We further extend this idea to fully extract the task information. To achieve this goal, we use a Neural Networks (NNs) as a task encoder. Specifically, this task encoder is implemented as a transformer encoder-like structure [35]. We first formulate the first i steps of collected trajectories $\tau_{0:i} = (s_0, a_0, r_0, \cdots, s_i, a_i, r_i)$ as a task specific information. The task trajectory $\tau_{0:i}$ is treated as a sequence of inputs to the task encoder. The output of the task encoder is a task embedding $e_{task} \in \mathbb{R}^d$, where d is the dimension of the embedding.

Then, we concatenate the task embedding and sequence embedding $e = [e_{task}; e_{seq}]$ and input them to the hyper-networks. Specifically, we define the hyper-network as a function of $f_\omega(\cdot)$ parameterized by ω. The output $\Theta = f_\omega(e)$ is a set of parameters for the memory

module.

According to the evaluation results in Table 6.8, the inclusion of a hyper-network in the DT-Memmodel improves generalization without the need for fine-tuning. However, it is worth noting that the hyper-network variant of DT-Mem(hyper-net) exhibits higher variance compared to DT-Mem. The primary reason for this higher variance is the uncertainty arising from the task information. In each run, different task-related sequences are collected, resulting in varying generated parameters for the memory module. Regarding the task fine-tuning results, we observe that the LoRA module outperforms other methods. This finding indicates that fine-tuning with LoRA enhances the model's adaptability. We hypothesize that the size of the hyper-networks model plays a role in these results. Fine-tuning a large model size (5.7M) with a small step-size (100k steps in our case) becomes challenging. In an effort to improve hyper-networks fine-tuning performance, we increased the fine-tuning dataset from 10k episodes to 30k episodes. These findings suggest that LoRA-based fine-tuning demonstrates better data efficiency.

6.5.9 LoRA hyper-parameters tuning

In this section, we explore the impact of LoRA hyper-parameters on the final fine-tuning results. LoRA employs three hyper-parameters: rank, lora_dropout, and lora_alpha. The rank parameter, denoted as m, determines the low-rank of adaptation matrices $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{A} \in \mathbb{R}^{m \times d}$, as described in Section 6.2.4. The lora_dropout refers to the dropout rate

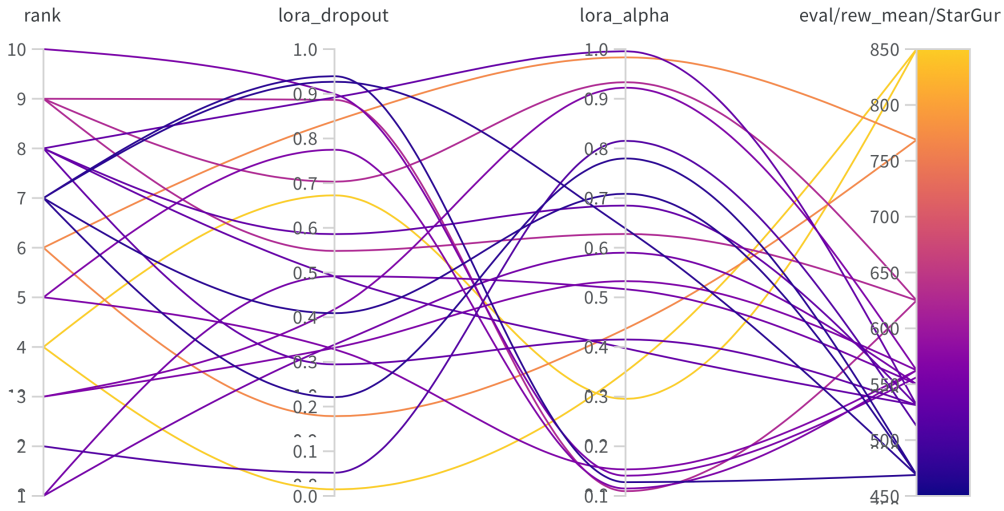


Figure 6.10: LoRA hyper-parameters tuning results.

applied to the LoRA neural networks, while lora_alpha controls the scaling factor of the LoRA outputs. Figure 6.10 presents the fine-tuning results, with the last column (**eval/rew_mean/StarGur**) specifically showcasing the fine-tuning results for the StarGunner game. To obtain the optimal set of parameters, we employ the Bayesian optimization method for parameter tuning, which suggests various parameter combinations that maximize the fine-tuning results.

Parameter	Importance score	Correlation score
rank	**0.486**	-0.132
lora_dropout	0.285	-0.561
lora_alpha	0.229	0.550

Table 6.9: Analysis of LoRA hyper-parameters

We further analyze these parameters and present the findings in Table 6.9. To gain

insights, we utilize two widely used metrics in the MLOps platform Weights&Biases[4].

Regarding the **importance score**, we train a random forest model with the hyper-parameters as inputs and the metric as the target output. We report the feature importance values derived from the random forest. This hyper-parameter importance panel disentangles complex interactions among highly correlated hyper-parameters. It facilitates fine-tuning of hyper-parameter searches by highlighting the hyper-parameters that significantly impact the prediction of model performance.

The **correlation score** represents the linear correlation between each hyper-parameter and the chosen metric (in this case, val_loss). A high correlation indicates that when the hyper-parameter has a higher value, the metric also tends to have higher values, and vice versa. Correlation is a useful metric, but it does not capture second-order interactions between inputs and can be challenging to compare when inputs have widely different ranges.

As shown in Table 6.9, rank emerges as the most important hyper-parameter that requires careful tuning. The correlation score of rank is -0.132, indicating that a smaller rank number leads to better fine-tuning results. Based on our findings, a rank value of 4 yields the best outcome. Lora_dropout and lora_alpha exhibit similar importance scores, suggesting that these two parameters can be treated equally. The correlation score reveals that a smaller lora_dropout value and a larger lora_alpha value result in improved performance.

[4]For better understanding, please refer to `https://docs.wandb.ai/guides/app/features/panels/parameter-importance?_gl=1*4s7cuj*_ga*MTQxNjYxODU0OC4xNjgzNjY4Nzg3*_ga_JH1SJHJQXJ*MTY4NDc5NDkzNS40MS4xLjE2ODQ3OTQ5NDIuNTMuMC4w`

6.6 Memory Module Visualization

Figure 6.11 illustrates the visualization of the memory module. Since memory operations are trained in conjunction with the transformer module, we select a later training episode at random to mitigate uncertainties regarding operational parameters. Due to time constraints, we trained on only two games simultaneously. In the revised version of the paper, we intend to provide visualizations for all games. For clearer visualization, we opted for a memory module of a smaller size, containing 128 memory slots.

Let's first discuss how memory modules update within the same game. As observed in the figure, for the Amidar game, the actively updated memory slots concentrate around rows 18, 84, and 117. This pattern is consistent across episodes, albeit with reduced activity. Such a trend indicates that during each training iteration, the transformer agent tends to overwrite the same memory slot contents. We noted a similar observation in the Assault game. Furthermore, we observed that the memory module's activity diminishes in later episodes. For instance, in the Assault game, the active memory slot in row 12 during episode 200k becomes less active by episode 201k. We hypothesize that as training progresses, the accumulated knowledge becomes sufficiently robust for retrieval, reducing the need for updates.

Moving on, when comparing the activity of memory slots across different games, there are intriguing overlaps. For instance, comparing Amidar 200k and Assault 200k reveals that memory slots around row 120 are active in both games. We surmise that this region retains

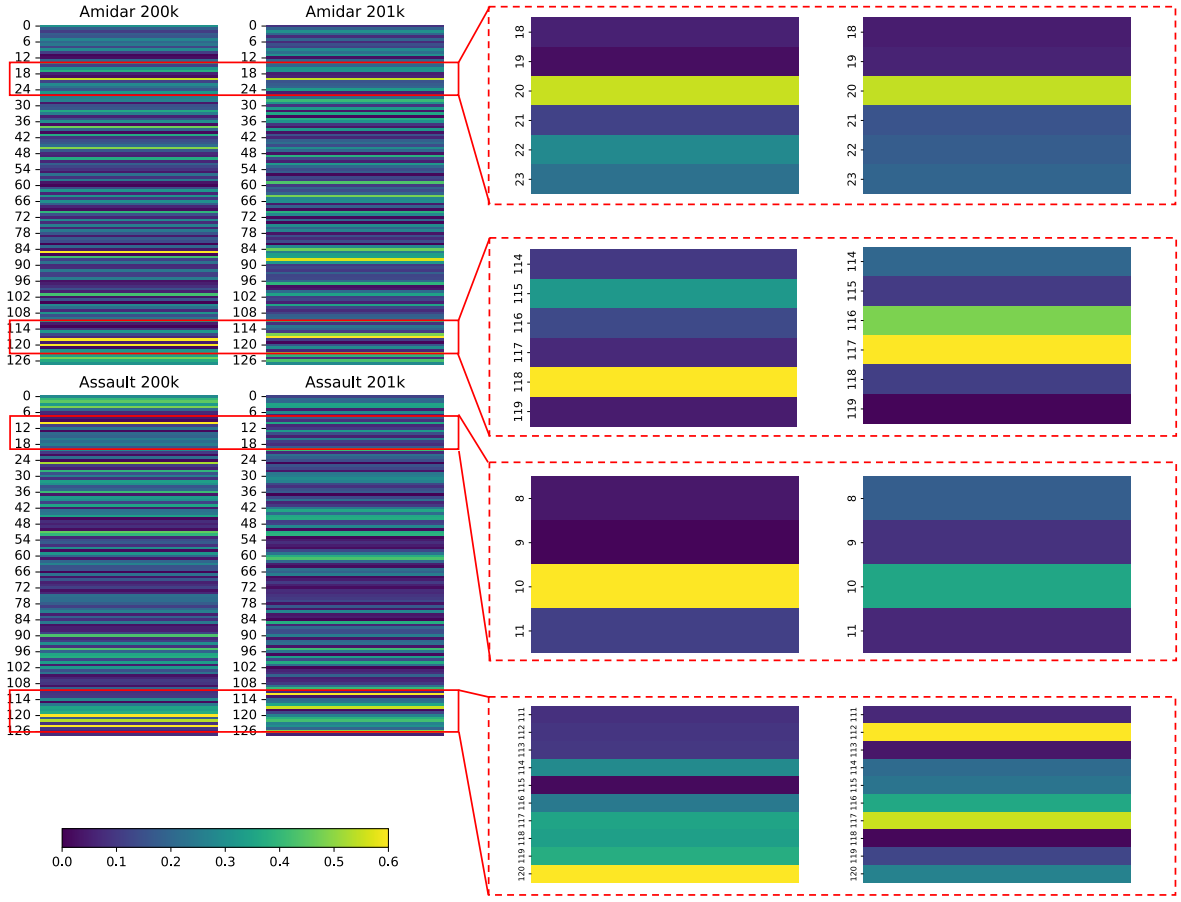


Figure 6.11: This visualization represents the memory module. In the figure, each row is derived from the mean of a vector that signifies a memory slot. Each depiction calculates the variation between two write operations in a single episode for each memory slot. Lighter shades indicate memory slots that have been actively updated post-write operations. The encircled areas highlight the comparison of active memory slots across different episodes.

cross-task knowledge shared between games. Additionally, the varying attention across other memory slots demonstrates how these slots assist the agent in decision-making across diverse games.

6.7 Summary

LLM-based RL algorithms have shown generalization across multiple tasks and games. We argue that this ability comes from implicit memory that fits a large number of parameters to the training data, which is inefficient in terms of model size. In contrast, we propose a new approach inspired by the concept of "working memory" called **D**ecision **T**ransformers with **M**emory (DT-Mem), which stores training experience explicitly in a content-addressable matrix module for later retrieval and use. Evaluation demonstrates that DT-Mem achieves better generalization on Atari games with only 10% of the model parameters compared to the state-of-the-art method. Furthermore, we demonstrate that fine-tuning DT-Memwith a small amount of data can produce state-of-the-art results on both Atari games and the Meta-World environment, when compared to MDT [15], PDT [19], and HDT [16].

Limitations The first limitation of our work is the sample efficiency of memory fine-tuning. The 10% fine-tuning dataset is still sizeable, and we plan to explore more sample-efficient methods in the future. We could for instance consider a setting with more tasks, each one with less data so that the inter-task generalization would be even more crucial to its performance. Additionally, this work does not propose a control strategy for collecting

data on a new task. For future work, we plan to investigate online data collection methods, which includes the design and learning of exploration strategies for an efficient fine-tuning on new tasks. Finally, the approach has been intuitively motivated, but it would be valuable to have a theoretical grounding that would show the structural limits of large models and how equipping them with a memory component overcomes them.

Societal Impact We do not foresee any significant societal impact resulting from our proposed method. The current algorithm is not designed to interact with humans, nor any realistic environment yet. If one chooses to extend our methods to such situations, caution should be exercised to ensure that any safety and ethical concerns are appropriately addressed. As our work is categorized in the offline-RL domain, it is feasible to supplement its training with a dataset that aligns with human intents and values. However, one must be wary that the way our architecture generalizes across tasks is still not well understood and as a consequence we cannot guarantee the generalization of its desirable features: performance, robustness, fairness, etc. By working towards methods that improve the computational efficiency of large models, we contribute to increase their access and reduce their ecological impact.

References

[1] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[2] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, OpenReview.net, 2021.

[4] H. Touvron, T. Lavril, G. Izacard, *et al.*, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023.

[5] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[6] A. Clark, D. de Las Casas, A. Guy, *et al.*, "Unified scaling laws for routed language models," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 4057–4086.

[7] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "The role of over-parametrization in generalization of neural networks," in *ICLR (Poster)*, OpenReview.net, 2019.

[8] A. Baddeley, "Working memory: Looking back and looking forward," *Nature reviews neuroscience*, vol. 4, no. 10, pp. 829–839, 2003.

[9] N. Cowan, "What are the differences between long-term, short-term, and working memory?" *Progress in brain research*, vol. 169, pp. 323–338, 2008.

[10] P. S. Goldman-Rakic, "Cellular basis of working memory," *Neuron*, vol. 14, no. 3, pp. 477–485, 1995.

[11] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[12] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *NIPS*, 2014, pp. 2204–2212.

[13] S. M. A. Eslami, N. Heess, T. Weber, *et al.*, "Attend, infer, repeat: Fast scene understanding with generative models," in *NIPS*, 2016, pp. 3225–3233.

[14] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," in *ICLR*, OpenReview.net, 2022.

[15] K. Lee, O. Nachum, M. Yang, *et al.*, "Multi-game decision transformers," in *NeurIPS*, 2022.

[16] M. Xu, Y. Lu, Y. Shen, S. Zhang, D. Zhao, and C. Gan, "Hyper-decision transformer for efficient online policy adaptation," *CoRR*, vol. abs/2304.08487, 2023.

[17] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, PMLR, 2016, pp. 1842–1850.

[18] S. Sukhbaatar, J. Weston, R. Fergus, *et al.*, "End-to-end memory networks," *Advances in neural information processing systems*, vol. 28, 2015.

[19] M. Xu, Y. Shen, S. Zhang, *et al.*, "Prompting decision transformer for few-shot policy generalization," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 24 631–24 645.

[20] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[21] P. C. Humphreys, A. Guez, O. Tieleman, L. Sifre, T. Weber, and T. P. Lillicrap, "Large-scale retrieval for reinforcement learning," in *NeurIPS*, 2022.

[22] S. Borgeaud, A. Mensch, J. Hoffmann, *et al.*, "Improving language models by retrieving from trillions of tokens," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 2206–2240.

[23] L. Chen, K. Lu, A. Rajeswaran, *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," in *NeurIPS*, 2021, pp. 15 084–15 097.

[24] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*, Association for Computational Linguistics, 2019, pp. 4171–4186.

[25] D. Brandfonbrener, A. Bietti, J. Buckman, R. Laroche, and J. Bruna, "When does return-conditioned supervised learning work for offline reinforcement learning?" In *NeurIPS*, 2022.

[26] T. Yu, D. Quillen, Z. He, *et al.*, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2019. arXiv: `1910.10897 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1910.10897`.

[27] D. Wu, J. Kang, Y. T. Xu, *et al.*, "Load balancing for communication networks via data-efficient deep reinforcement learning," in *GLOBECOM*, IEEE, 2021, pp. 1–7.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[29] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1582–1591.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 1856–1865.

[31] D. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Comput.*, vol. 3, no. 1, pp. 88–97, 1991.

[32] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[33] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 104–114.

[34] J. von Oswald, C. Henning, J. Sacramento, and B. F. Grewe, "Continual learning with hypernetworks," in *ICLR*, OpenReview.net, 2020.

[35] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

Chapter 7

Hierarchical Enhanced Safety Actions

In Chapter 6, we explored the extensive applications of our models, showcasing their adaptability and impact across various domains. The insights garnered from these applications underscore the practical relevance of our research. Chapter 7 will take us into the realm of enhanced safety actions in hierarchical models. This chapter aims to address some of the critical challenges and ethical considerations associated with the deployment of these models in real-world scenarios, marking a crucial step in our journey towards responsible and effective AI systems.

7.1 Introduction

Cellular communications have penetrated to every corner of our daily lives. To support our ever-increasing communication demands, cells have been deployed across the territory to

provide better services [1]. However, due to regulatory and engineering constraints, cells can not be deployed arbitrarily [2]. This leads to a mismatch between the relatively uniform distribution of cells and the uneven demographic distribution of people. As a result, a cellular system usually witnesses highly imbalanced loads across different cells, resulting in unsatisfied users and wasted resources.

Extensive efforts have been made to balance the load by migrating User Equipment (UEs) across cells [3]–[5]. The existing Load Balancing (LB) methods can be divided into two categories, the Active-UE LB (AULB) and the Idle-UE LB (IULB). The AULB methods utilize the Handover (HO) mechanism to offload active mode UEs (i.e., UEs currently transceiving signals) from busy serving cells to less-busy neighboring cells [6], [7]. Such methods achieve instantaneous load balancing results, by paying the price of increased system overhead. The IULB methods leverage the Cell Re-selection (CR) mechanism to move idle mode UEs (i.e., UEs connected but not transceiving signals) from congested camping cells to other cells [8], [9]. These methods are more lightweight, since CR requires less system overhead than HO. Yet, the benefit is realized only after the migrated idle UEs become active.

Challenge: The actions of AULB may conflict with the actions of IULB (and vice versa), resulting in unexpected degradation on system performance and safety issues.

In order to overcome this challenge, in this chapter, we adopt a **H**ierarchical **P**olicy **L**earning (HPL) method, which integrates both AULB and IULB into a two-level

Reinforcement Learning (RL) structure. Concretely, the upper level adjusts AULB actions, and the lower level controls the IULB actions. The upper level aims to optimize the system performance directly as an RL reward, and at the same time, learns to set a subgoal for the lower level. This subgoal is a desired RL state, which further improves the upper-level reward (and yet cannot be achieved with only upper-level actions). By approaching this subgoal, the lower level 1) indirectly enhances the system performance, and 2) is enforced to align with the upper level. In this way, a collaboration is established between AULB and IULB, eliminating the potential conflicts.

The major contribution is summarized as follow.

Contribution: this chapter develops HPL - the first hierarchical learning method that integrates different LB mechanisms (i.e., AULB and IULB) in a collaborative way.

We evaluate the proposed HPL method against the State-Of-The-Art (SOTA) RL-based LB methods in a system-level network simulator. The simulation results show that, under different UE density settings, our HPL method always outperforms the SOTA methods. Specifically, compared to a direct combination of SOTA AULB and IULB, HPL improves the average throughput by up to 24.1%, while reducing the standard deviation of throughput by up to 31.0%.

7.2 Preliminaries

7.2.1 Cellular Network Terminologies

For the sake of clarity, we define some cellular network terminologies. We use the term "load" to refer to the number of UEs being served. We use a Base Station (**BS**) to describe a physical site, where radio access devices are placed. Consider a cellular network with N_B BSs, each of which consists of N_S non-overlapping **sectors**. A sector is serving the UEs located on a certain direction of its hosting BS. A sector supports N_C carrier frequencies, each of which corresponds to a **cell**. A cell is a service entity serving the UEs within a certain direction of a BS and on a certain carrier frequency.

7.2.2 Performance Metrics

Suppose there are N_U UEs (either active or idle) in the network. Define U_i as the set of UEs associated with the i-th cell. Among U_i, there are both active UEs (denoted as U_i^a) and idle UEs (denoted as U_i^d). Naturally, we have $U_i = U_i^a \cup U_i^d$. Further, let $u_{i,k}$ denote the k-th UE in the i-th cell. Note that an idle UE at the current moment may become active in the future, and vice versa. We aim to balance the assignments of UEs to different cells, so as to enhance the following metrics.

The first metric is the average throughput G_{aver}, i.e.,

$$G_{aver} = \frac{1}{N_U} \sum_i \sum_k \frac{A_{i,k}}{T}, \tag{7.1}$$

where T is the time period of interest, and $A_{i,k}$ is the total size of packets received by $u_{i,k}$ within T. Improving this metric means to increase the overall system performance.

The second metric is the minimum throughput G_{min}, i.e.,

$$G_{min} = \min_{i,k} \left(\frac{A_{i,k}}{T} \right), \tag{7.2}$$

which captures the worst-case UE performance.

Last but not least, we consider the reciprocal of the Standard Deviation (SD) of throughput as the third metric G_{sd} , i.e.,

$$G_{sd} = \left(\sqrt{\frac{1}{N_U} \sum_i \sum_k (\frac{A_{i,k}}{T} - G_{aver})^2} \right)^{-1}. \tag{7.3}$$

Maximizing this metric reduces the gap between different UEs' performance, and thus provides fairer services to all UEs.

7.2.3 Active UE Load Balancing (AULB) via Handover

The first category of LB methods rely on the HO of active UEs. For the sake of generality, we consider a common Reference Signal Received Power (RSRP[1]) based HO mechanism, which can cover different variants, such as Cell Individual Offset (CIO) based HO or A2/A5 event based HO in LTE/5G networks. Concretely, every UE compares the RSRP value of its serving cell against the values of its neighboring cells. If the following condition holds, then the active UE will be handed over to a neighboring cell, i.e.,

$$RSRP_j > RSRP_i + \alpha_{i,j} + H, \tag{7.4}$$

where $RSRP_i$ represents the UE's RSRP from the serving cell i, $RSRP_j$ denotes the UE's RSRP from a neighboring cell j, $\alpha_{i,j}$ is the HO threshold from cell i to cell j, and H is the HO hysteresis. This HO threshold $\alpha_{i,j}$ is a pair-wise directional variable (e.g., $\alpha_{i,j} \neq \alpha_{j,i}$). By changing $\{\alpha_{i,j}\}$, we are able to adjust the HO boundaries between cells, and therefore balance the numbers of active UEs across cells.

7.2.4 Idle UE Load Balancing (IULB) via Cell Re-selection

Another category of LB methods depend on the Cell Re-selection (CR) of idle UEs. When a UE is turned on, it first enters the idle mode and "camps" on a cell. An idle UE is ready

[1]This variable can be Reference Signal Received Quality (RSRQ) as well.

to initiate a potential dedicated service or to receive a broadcast service. Once becoming active, the UE usually will stay in the same cell, where it camped during the idle mode.

An idle UE can camp on another cell via the CR procedure, so as to stay connected when moving. This CR procedure will be triggered, if the following condition holds for an idle UE:

$$RSRP_i < \beta_{i,j}, \text{ and } RSRP_j > \gamma_{i,j}, \tag{7.5}$$

where $\beta_{i,j}$ and $\gamma_{i,j}$ are pairwise and directional RSRP thresholds to trigger CR from a camping cell i to a neighboring cell j. Again, the CR mechanism represented by condition (7.5) is a generalized one[2]. By adjusting $\{\beta_{i,j}\}$ and $\{\gamma_{i,j}\}$, we can achieve a balanced distribution of idle UEs across cells. This helps reduce the congestion when idle UEs become active.

7.3 The Problem and The Challenge

7.3.1 The Hybrid Load Balancing Problem

In this chapter, we aim to solve a hybrid LB problem, where both AULB and IULB are applied to achieve balanced load and better system performance. Formally, we define this hybrid LB problem as follows.

$$\max_{\{\alpha_{i,j}\},\{\beta_{i,j}\},\{\gamma_{i,j}\}} G, \tag{7.6}$$

[2]The current format follows LTE/5G's lower priority inter-frequency CR. Also, $\beta_{i,j} = 0$ provides LTE/5G's higher priority inter-frequency CR.

$$\text{s.t. } \alpha_{i,j} \in [\alpha_{min}, \alpha_{max}], \tag{7.7}$$

$$\beta_{i,j} \in [\beta_{min}, \beta_{max}], \tag{7.8}$$

$$\gamma_{i,j} \in [\gamma_{min}, \gamma_{max}], \tag{7.9}$$

where G is the system performance, α_{min} and α_{max} define the controllable range of AULB actions, and β_{min}, β_{max}, γ_{min} and γ_{max} define the controllable range of IULB actions. In this chapter, we use three different type of metrics (G_{aver}, G_{min}, and G_{sd}) to measure system performance G.

7.3.2 Potential Conflicts Between AULB and IULB

Although individual AULB and IULB work quite well respectively, it is non-trivial to fuse them together, mainly due to the potential conflicts between these two methods (i.e., the **Challenge** stated in Section 7.1).

A motivating example is presented as follows. Consider two co-located cells, cell 1 and cell 2, residing on different carrier frequencies. Support an AULB method sets $\alpha_{1,2} = \alpha_{2,1} = 2dB$ and $H = 1dB$, while an IULB method sets $\beta_{1,2} = -100dB$ and $\gamma_{1,2,} = -106dB$[3]. A UE is now idle, and camps on cell 1 with $RSRP_1 = -101dB$ and $RSRP_2 = -105dB$. As one can see, the CR condition (7.5) is satisfied, i.e., $RSRP_1 < \beta_{1,2}$ and $RSRP_2 > \gamma_{1,2,}$. Hence, this UE re-selects cell 2 and camps on it. Let's say, immediately after that, this UE

[3]A system may set $\gamma_{i,j} < \beta_{i,j}$ and $\gamma_{j,i} > \beta_{j,i}$, so that UEs can be migrated towards more preferable cells, e.g., cells with larger bandwidth.

becomes active, and uses cell 2 as its serving cell. As this moment, the UE finds out that the HO condition (7.4) from cell 2 to cell 1 holds now, i.e., $RSRP_1 > RSRP_2 + \alpha_{2,1} + H$. Consequently, this UE is moved back to cell 1 via HO. Such UE oscillations among cells could lead to degraded performance and wasted resources (evidence to be presented in Section 7.5).

7.4 Hierarchical Policy Learning

To overcome the aforementioned challenge, in this section, we propose our Hierarchical Policy Learning (HPL) method.

7.4.1 Markov Decision Process Modeling

We adopt the (deep) RL framework, which has been proven as effective and efficiency for LB problems [5], [7]. The first step of applying RL is to formulate the hybrid LB problem as a Markov Decision Process (MDP). This MDP is defined as a tuple (S, A, R, P) as follows:

- S**:** is the state space $s \in \mathbb{R}^{12}$, which comprises the number of active UEs in each cell $s_{ue} \in \mathbb{R}^4$, bandwidth utilization in each cell $s_{band} \in \mathbb{R}^4$, and average throughput in each cell $s_{tput} \in \mathbb{R}^4$.

- A**:** is the action space. Every action contains two parts. The first part a^H corresponds to the HO parameters that control AULB actions (i.e., $\alpha_{i,j}$). The second part a^L corresponds to the CR parameters that control IULB actions (i.e., $\beta_{i,j}$ and $\gamma_{i,j}$).

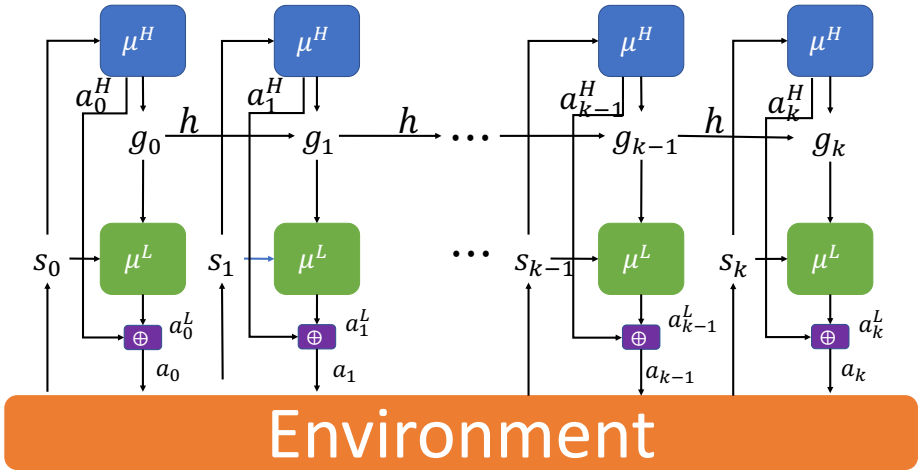


Figure 7.1: The Architecture of Hierarchical Policy Learning

- R**:** is the reward (to be defined in Section 7.4.3).
- P**:** is the transition probability function.

7.4.2 Hierarchical RL Structure

To solve the conflicts between AULB and IULB, we propose a two-level hierarchical policy learning structure, which is shown in Figure 7.1. Basically, the higher level controls the AULB actions a^H with policy μ^H, and the lower level controls the IULB actions a^L with policy μ^L. The actions a^H and a^L are HO and CR parameter adjustments, respectively. The resulting system performance is then collected as the RL reward.

At every time step t, both higher-level and lower-level policies receive a state s_t from the environment. Based on this state, the higher-level policy $\mu^H(s_t)$ produces a higher-level

control action a^H. This higher-level action is used in two ways. 1) It is fed to the system to HO control. 2) This higher-level action is also used to produce the subgoal for the lower level. The subgoal is denoted as $g_t \in \mathbb{R}^d$, where d is the dimension of subgoal and t is the timestep. It is generated by the goal transition function $g_t = f(s_t, a_t^H)$. In other words, every time step, this function generates a subgoal according to the current state s_t and higher-level action a_t^H. We utilize a LSTM[10] network to implement our subgoal transition function, i.e., $g_t = LSTM(s_t, a_t^H)$. The use of LSTM makes sure that the current generated goal is consistent with the previous goals.

Based on the current state s_t and the subgoal g_t, the lower-level policy $\mu^L(s_t, g_t)$ produces an IULB action a_t^L. Since the subgoal embeds the higher-level actions, the lower-level policy is forced to be aligned with the higher-level policy when achieving this subgoal. The combined higher and lower actions $a_t = a_t^H \oplus a_t^L$ is applied to the system, so that the environment can return the next state s_{t+1} and reward r_t.

In our hierarchical reinforcement learning framework, the LSTM (Long Short-Term Memory) network plays a pivotal role in generating subgoals. The architecture of this LSTM network is designed to capture the complex dynamics of subgoal transitions, which are integral to the system's overall performance.

The LSTM network comprises several layers, each designed to process temporal sequences of data effectively. Specifically, the architecture includes:

- **Input Layer:** Receives the current state s_t and the higher-level action a_{Ht} as input.

- **Hidden Layers:** Multiple LSTM layers, each consisting of a certain number of LSTM units. These layers are responsible for capturing the temporal dependencies and nonlinearities in the sequence of states and actions.

- **Output Layer:** Produces the subgoal g_t for the next timestep. The output dimension is determined by the dimensionality of the subgoals.

- **Loss Function:** The training of the LSTM network is guided by a loss function, specifically designed to optimize the subgoal generation. This is expressed as $L_{\text{generator}} = -Q_{\mu_L}(s_t, a_t, g_t)$, where Q_{μ_L} represents the advantage value function.

This architecture ensures that the generated subgoals are consistent with the system's current state and the objectives set by the higher-level policy. The LSTM's ability to maintain a memory of past states allows for a smoother transition of subgoals over time, essential for the stability and effectiveness of the hierarchical learning process.

The specific configuration details, such as the number of LSTM units in each layer and the total number of layers, should be tuned according to the complexity of the task and the dimensional requirements of the state and action spaces.

By detailing the architecture of the LSTM network used in our subgoal generation process, we aim to provide clarity and enhance the reproducibility of our approach for future research endeavors.

he LSTM network's functionality can be mathematically represented as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{7.10}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{7.11}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{7.12}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{7.13}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7.14}$$

$$h_t = o_t * \tanh(C_t) \tag{7.15}$$

where x_t is the input at timestep t, h_t is the hidden state, C_t is the cell state, f_t, i_t, o_t are the forget, input, and output gates, respectively, and W and b are the weights and biases for each gate.

The following pseudocode outlines the procedure of subgoal generation using the LSTM network:

Algorithm 5: Subgoal generation

```
Initialize LSTM network with weights and biases;
for for each timestep t do
    Receive current state s_t and higher-level action a_Ht;
    Concatenate s_t and a_Ht to form input x_t;
    Pass x_t through LSTM network;
    Compute new cell state C_t and hidden state h_t;
    Output the generated subgoal g_t;
```

This pseudocode and mathematical formulation provide a detailed view of the internal

workings of the LSTM network in our subgoal generation process. By understanding these operational details, researchers and practitioners can replicate and build upon our methodology more effectively.

7.4.3 Reward Optimization for Conflicting Actions

The reward for each level is different. At time step $t+1$, the higher-level policy receives the reward r_t directly from the environment, i.e., $r_t^H = r_t$, which is a system performance metric (e.g., average throughput). The lower-level reward r_t^L evaluates whether the subgoal has been achieved, and is calculated by a reward function $r_t^L = \eta(g_t, s_{t+1})$.

A subgoal is defined as a goal state that is expected to provide a larger higher-level reward than the current state. Usually, a goal state is not achievable with only the higher-level actions. Hence, the lower level comes in to play, so that the system performance could be further improved. Accordingly, we define the lower-level reward function based on the distance between the current state and the goal state, i.e.,

$$r_t^L = \eta(g_t, s_{t+1}) = -||\phi(g_t) - \phi(s_{t+1})||_2, \tag{7.16}$$

where $\phi(\cdot)$ is an embedding function to map a high-dimensional space to a low-dimensional space, so that we can use the low-dimensional euclidean distance to describes how close two high-dimensional states are. The lower-level policy is rewarded for taking actions that yield

states s_{t+1} close to the desired subgoal g_t.

Both policies can be trained using advanced RL methods, by incorporating g_t as an additional input into the value and policy functions. In this chapter, we choose the state-of-the-art on-policy learning method Proximal Policy Optimization (PPO) [11] as our policy training method, due to its robustness.

Given the lower-level reward in Eqn. (7.16), the lower-level Q-value function is to minimize the loss:

$$\begin{aligned} L(\mu^L, D) =& \mathbb{E}_{(s_t,a_t,g_t,r_t,s_{t+1},a_{t+1},g_{t+1})\sim D}[(Q_{\mu^L}(s_t, a_t, g_t) \\ & - r(g_t, s_{t+1}) - \gamma Q_{\mu^L}(s_{t+1}, a_{t+1}, g_{t+1})], \end{aligned} \tag{7.17}$$

where Q_{μ^L} is the advantage value function of the lower level, and D is the replay buffer. This lower-level loss enforces that the learned actions should move the state close to the subgoal.

The higher-level reward function is shown in Eqn. (7.18). The learned policy aims to maximize the future collective rewards based on the current state. In other words, the higher-level policy generates a subgoal that is expected to improve the system performance (which is our major objective).

$$\begin{aligned} L(\mu^H, D) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},a_{t+1})\sim D}[(Q_{\mu^H}(s_t, a_t) - r - \\ \gamma Q_{\mu^H}(s_{t+1}, a_{t+1})], \end{aligned} \tag{7.18}$$

where Q_{μ^H} is the advantage value function of the higher level.

Combining the proposed two policies, the learned actions for hybrid AULB and IULB work collaboratively in terms of improving the system performance without conflict with each other. The higher-level AULB policy takes the major step towards the optimal system performance by choosing it's own actions as well as setting the subgoal for the lower-level IULB policy. By fulfilling the subgoal, the lower-level policy helps further improve the system performance upon what has been achieved by the higher level.

7.4.4 Subgoal Generation

Recent advances on Hierarchical RL (HRL) employ the higher-level policy to generate the subgoal directly. Different from them, in our proposed HRL method, we employ a Long-Short Term Memory (LSTM) Neural Network (NN) as our goal generator. The benefit

Algorithm 6: Hierarchical Policy Learning Procedure

Randomly initiate a_0 and g_0;
for *every time step* t $(t = 1, 2, ...)$ **do**
  Apply action a_{t-1} to the environment;
  Collect the current state s_t and the reward r_{t-1} from the environment;
  Calculate the rewards of both levels, i.e.,$r^L_{t-1} = R(g_{t-1}, s_t)$ and $r^H_{t-1} = r_{t-1}$;
  Compute the advantage functions Q_{μ^L} and Q_{μ^H};
  Update the parameters of the lower-level policy μ_L by minimizing the loss presented in Eqn. (7.17);
  Update the parameters of the higher-level policy μ_H by minimizing the loss presented in Eqn. (7.18);
  Update the LSTM parameters f_{LSTM} of the subgoal generator by minimizing the loss presented in Eqn. (7.19);
  Generate a higher-level action a^H_t with policy $\mu_H(s_t)$;
  Use the subgoal generator to produce a new subgoal $g_t = f_{LSTM}(s_t, a^H_t)$;
  Generate a lower-level action a^L_t with policy $\mu_L(s_t, g_t)$;
  Concatenate actions from both levels to generate the united action $a_t = a^H_t \oplus a^L_t$;

for this design is two-fold. First of all, a LSTM NN can approximate a non-linear high-dimensional goal transition function, which is not achievable with existing two-dimensional goal-transition function [12].. Second, when generating the subgoals, it is important to maintain a certain level of consistency between the current subgoal and the previous ones. An LSTM NN recursively computes the hidden state with the previous states being taken into consideration, and therefore generates subgoals smoothly.

We need to train this LSTM NN to generate g_t that can further improve r_t. To this end, the training loss of this LSTM NN is set as the opposite of advantage value function (note that the advantage value function captures the increment in the reward), i.e.,

$$L_{generator} = -Q_{\mu^L}(s_t, a_t, g_t). \tag{7.19}$$

By minimizing this loss, the LSTM NN is trained to produce a goal state g_t that can further improve r_t. This LSTM-based goal generator is trained together with control policies.

7.4.5 The Overall HPL Procedure

The whole HPL procedure is summarized as **Procedure 6**.

7.5 Evaluation

7.5.1 Experiment Setup

The experiments reported here utilizes a proprietary system-level network simulator. This simulator is designed for emulate 4G/5G communication network behaviors. The simulation scenario is presented in Figure 7.2. There are in total 7 BSs, each of which supports 3 sectors. In each sector, there are 4 cells residing on 4 different carrier frequencies, respectively. (These 4 carrier frequencies are identical across different sectors and BSs). The scenario is wrapped around at the edges. We emulate different UE density settings, by setting the average number of UEs per cell to 10, 20, and 30, respectively. Theses UEs are uniformly distributed geographically at initialization. The UE movement follows a random walk process with an average speed of $3m/s$. The packet arrival follows a Poisson process with an average inter-arrival time of $200ms$.

7.5.2 Methods Evaluated

- **AULB** is trained to control AULB only with one-level PPO. The IULB actions are set to the default values. This emulates the SOTA deep RL based mobility load balancing methods (e.g., [7], [13], [14]).
- **IULB** is trained to control IULB only with one-level PPO. The AULB actions are set to the default values. This serves as the representative of the SOTA cell re-selection

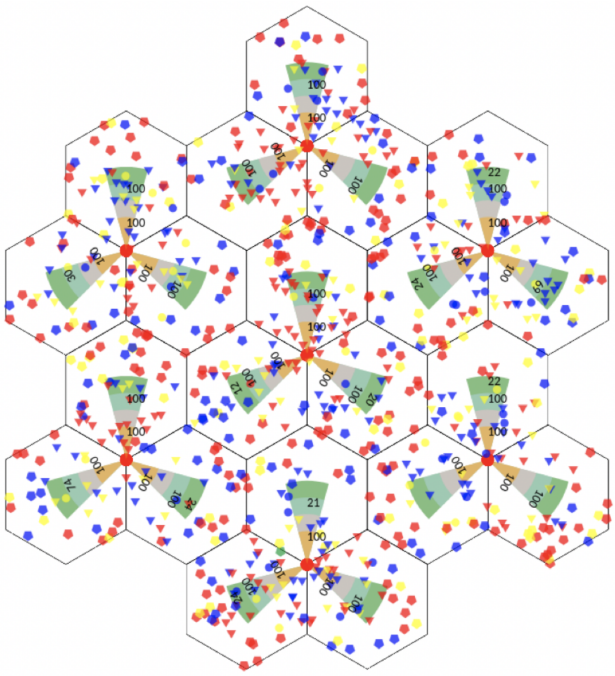


Figure 7.2: The simulation scenario.

methods (e.g., [8]).

- **Sequential** It produces AULB and IULB actions separately using the above two methods, and then directly combines two kinds of actions together as one action.
- **AULB+IULB** is trained to control both IULB and AULB simultaneously with one-level PPO.
- **HPL** is trained to control both IULB and AULB at the same time with the proposed HPL method.

7.5.3 Performance Comparison

To evaluate whether the system load is balanced, we first present the Standard Deviation (SD) of throughput across different cells in Figure 7.3. The lower the SD of throughput is, the more balanced the load is. From Figure 7.3, we can observe that the proposed HPL algorithm consistently outperforms other baselines methods in terms of the balance of load. In the most dense scenario, compared to AULB, IULB, Sequential, and AULB+IULB, HPL reduces the SD of throughput by 23.1%, 28.6%, 20.2% and 31.0%, respectively. We also notice that with the increasing number of UEs, the performance of AULB+IULB decreases quickly. It achieves much higher SD of throughput than AULB or IULB alone. This phenomenon indicates that the conflicts between the IULB and AULB increase with the number of UEs. It also shows that the proposed HPL is able to resolve the conflicts between IULB and AULB, and thus better balances the load.

The balanced load does not necessarily lead to better services, as a low SD of throughput could imply either equally good services or evenly bad services. Therefore, we further evaluate the average throughput to show that our HPL method can provide better service while keeping the load balanced. In Table 7.1, we presented the average throughput of different methods. We can see that the proposed HPL achieves the best performance among all. In the most dense scenario, compared to AULB, IULB, Sequential, and AULB+IULB, HPL improves performance by 13.0%, 13.9%, 32.3%, and 24.2%, respectively. Specifically, compared to the joint RL of AULB+IULB, HPL increases the

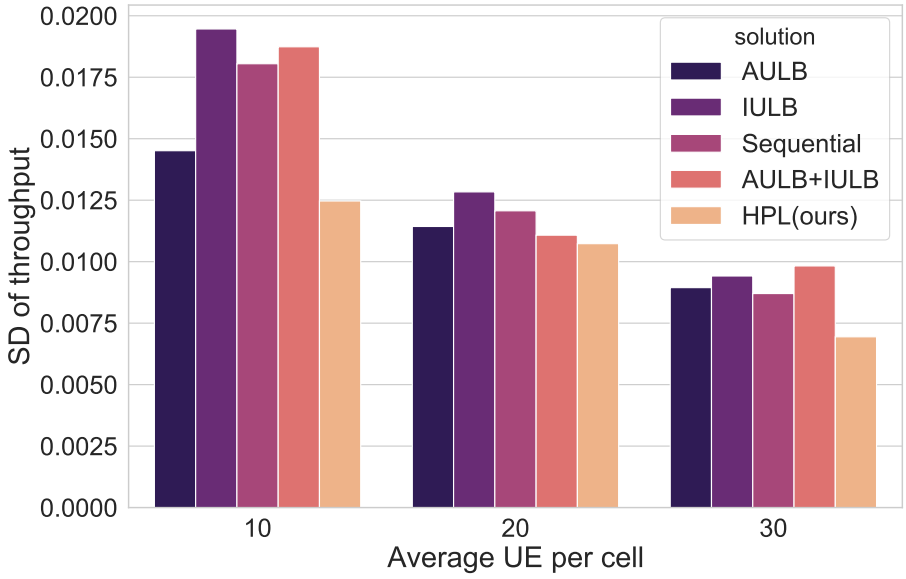


Figure 7.3: The SD of throughput of different methods (the lower the better).

average throughput by 0.3%, 18.9%, and 24.1%, respectively under three UE densities.

There is one interesting finding: although Sequential and AULB+IULB both combine two LB mechanisms, they achieve lower throughput than methods using only one individual LB mechanisms. This is another evidence that different LB mechanisms can conflict with each other, leading to large degradation of system performance. By applying our HPL method, we are able to reduce these conflicts and thus achieve better performance than all SOTA methods.

While average throughput measures the overall performance, we further analyze the minimum throughput that reflects the worst-case performance of individual UEs. In

[4]The percentage shows the improvement over AULB+IULB.

	Average UEs per cell		
Method	10	20	30
AULB	0.593	0.231	0.123
IULB	0.491	0.199	0.122
Sequential	0.400	0.103	0.105
AULB+IULB	0.595	0.201	0.112
HPL[4]	**0.597 (+0.3%)**	**0.239 (+18.9%)**	**0.139 (+24.1%)**

Table 7.1: The average throughput (Mbps) of different methods

Table. 7.2, we present minimum throughput achieved by different methods. From this table, we note that HPL outperforms all SOTA methods in terms of the worst-case throughput across UE density settings. Compared to AULB+IULB, HPL increases the minimum throughput by 0.2%, 0.176%, and 13.6%, respectively under three UE densities. This result again confirms the advantage of our hierarchical RL structure, comparing to the straightforward joint AULB+IULB method.

	Average UEs per cell		
Method	10	20	30
AULB	0.400	0.161	0.085
IULB	0.402	0.156	0.079
Sequential	0.392	0.150	0.073
AULB+IULB	0.415	0.158	0.081
HPL	**0.416 (+0.2%)**	**0.176 (+11.4%)**	**0.092 (+13.6%)**

Table 7.2: The minimum throughput (Mbps) of different methods

Finally, we evaluate the system overhead in terms of UE HO counts. Table 7.3 presents the averaged number of HO per cell within a hour. From this table, we observe that HPL results in the smallest number of HO in all UE density settings. More precisely, compared to

AULB+IULB, HPL decreases the number of handovers by 5.5%, 9.9%, and 2.2%, respectively under three UE densities. This suggests that, by resolving the conflicts between AULB and IULB, the proposed HPL method avoids some meaningless or even harmful HO operations, and thus overcomes the aforementioned challenge.

	Average UEs per Cell		
Method	10	20	30
AULB	8.82	39.86	99.59
IULB	6.61	35.04	102.48
Sequential	6.91	34.21	107.20
AULB+IULB	5.86	34.28	102.83
HPL	**5.54 (-5.5%)**	**30.90 (-9.9%)**	**98.52 (-2.2%)**

Table 7.3: The number of handovers per cell per hour of different methods (the lower the better)

7.6 Summary

In this chapter, we study a hybrid communication LB problem, where both active and idle UEs can be migrated across cells for better system performance. A major challenge lies in the conflicts between active UE LB and idle UE LB mechanisms. To conquer this challenge, we propose a two-level Hierarchical Policy Learning (HPL) method. HPL coordinates AULB and IULB, by setting an AULB-determined subgoal for IULB to accomplish. To the best of our knowledge, this is the first hierarchical learning structure for the hybrid communication LB problem. System-level simulations demonstrate HPL's significant improvements over the SOTA methods on three key performance metrics.

References

[1] L. Chiaraviglio, G. Bianchi, N. Blefari-Melazzi, and M. Fiore, "Will the proliferation of 5g base stations increase the radio-frequency "pollution"?" In *VTC Spring*, IEEE, 2020, pp. 1–7.

[2] M. Dong, T. Kim, J. Wu, and E. W. M. Wong, "Cost-efficient millimeter wave base station deployment in manhattan-type geometry," *IEEE Access*, vol. 7, pp. 149 959–149 970, 2019.

[3] M. M. Hasan, S. Kwon, and J. Na, "Adaptive mobility load balancing algorithm for LTE small-cell networks," *IEEE Trans. Wirel. Commun.*, vol. 17, no. 4, pp. 2205–2217, 2018.

[4] H. Zhang, X. Qiu, L. Meng, and X. Zhang, "Design of distributed and autonomic load balancing for self-organization LTE," in *Proceedings of the 72nd IEEE Vehicular Technology Conference, VTC Fall 2010, 6-9 September 2010, Ottawa, Canada*, IEEE, 2010, pp. 1–5. DOI: 10.1109/VETECF.2010.5594567. [Online]. Available: https://doi.org/10.1109/VETECF.2010.5594567.

[5] P. M. Luengo, R. Barco, J. M. Ruiz-Avilés, I. de la Bandera, and A. Aguilar, "Fuzzy rule-based reinforcement learning for load balancing techniques in enterprise LTE femtocells," *IEEE Trans. Veh. Technol.*, vol. 62, no. 5, pp. 1962–1973, 2013. DOI:

10 . 1109 / TVT . 2012 . 2234156. [Online]. Available: https://doi.org/10.1109/TVT.2012.2234156.

[6] S. He, T. Wang, and S. Wang, "Qos-aware load balancing in dense cellular networks with dynamic user traffic," in *ICC*, IEEE, 2018, pp. 1–6.

[7] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Deep reinforcement learning based mobility load balancing under multiple behavior policies," in *ICC*, IEEE, 2019, pp. 1–6.

[8] Mingju Li, Xiaoming She, and Lan Chen, "Access probability aware cell reselection for load balancing," in *2009 IEEE International Conference on Communications Technology and Applications*, 2009, pp. 106–109. DOI: 10.1109/ICCOMTA.2009.5349229.

[9] J. Park, "Ue-initiated cell reselection game for cell load balancing in a wireless network," *Wirel. Commun. Mob. Comput.*, vol. 2018, 2018.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[13] P. H. Barros, I. Cardoso-Pereira, L. Foschini, A. Corradi, and H. S. Ramos, "Load balancing in D2D networks using reinforcement learning," in *2019 IEEE Symposium on Computers and Communications, ISCC 2019, Barcelona, Spain, June 29 - July 3, 2019*, IEEE, 2019, pp. 1–6. DOI: 10.1109/ISCC47284.2019.8969767. [Online]. Available: https://doi.org/10.1109/ISCC47284.2019.8969767.

[14] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-uav-enabled load-balance mobile-edge computing for iot networks," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6908, 2020.

Chapter 8

Conclusion and Future Work

This chapter brings the thesis to a close. In Section 8.1, we conclude the significant contributions of our work. Section 8.2 provides a comprehensive summary of the key findings and insights derived from this research. Finally, Section 8.3 outlines various promising avenues for future research, building upon the groundwork laid by this thesis.

8.1 Conclusion

Throughout this thesis, we have embarked on a comprehensive exploration of the challenges and innovations in applying deep reinforcement learning (DRL) to real-world applications. Focusing on critical areas such as data efficiency, training complexity, model generalization, and safety concerns, this work has not only addressed existing gaps but also set forth a new paradigm in the application of DRL.

In the quest to enhance data efficiency, as detailed in Chapter 4, we introduced a multi-teacher knowledge distillation approach. This novel strategy leveraged the strengths of various system models, each representing unique traffic patterns, to facilitate a more efficient and robust learning process in model-based reinforcement learning. The application of knowledge distillation techniques has been a game-changer, significantly reducing the required volume of training data while simultaneously enhancing the model's generalization capabilities across diverse scenarios. This breakthrough is a testament to the synergy that can be achieved by integrating concepts from different realms of machine learning.

Addressing the daunting challenge of training complexity, Chapter 5 presented a groundbreaking automatic curriculum learning framework. By employing a hyper-net to parameterize a network of curricula, this approach has demonstrated remarkable proficiency in streamlining the training process. This methodology is particularly potent in the context of robotic manipulation tasks, where it has shown to not only accelerate training efficiency but also elevate final performance outcomes, thereby marking a significant advancement in the field.

In tackling the perennial issue of model generalization, as elucidated in Chapter 6, our research made a pioneering leap by integrating an internal working memory module. This innovation, featuring a memory matrix capable of sophisticated information processing, has propelled our models to new heights of adaptability and effectiveness. The ensuing

improvements in training efficiency and task versatility across various domains – from gaming to object manipulation – underscore the transformative potential of this approach.

Chapter 7 addressed the intricate safety concerns inherent in network load-balancing problems. The development and implementation of the Hierarchical Policy Learning (HPL) framework marks a significant stride in this domain. By orchestrating a two-tiered learning structure, the HPL framework has shown remarkable adeptness in coordinating actions and mitigating conflicts, thus enhancing both system performance and safety. The efficacy of this framework in reconciling multi-level actions and addressing complex safety challenges paves the way for safer and more reliable DRL applications.

In conclusion, this thesis stands as a testament to the notion that while the challenges of applying DRL in real-world scenarios are formidable, they are far from insurmountable. The methodologies and insights presented in this work constitute a significant leap forward in enhancing the efficiency, generalization, and safety of DRL applications. The implications of these advancements are far-reaching, extending well beyond the academic sphere into practical realms spanning networking, robotics, and other industries. It is with great optimism that we anticipate these contributions to act as a catalyst for future explorations and innovations in deep reinforcement learning, ultimately unlocking its full potential in a plethora of real-world applications.

8.2 Synthesis of Key Findings

This section synthesizes the findings about the thesis, reflecting on the journey from the fundamental concepts of deep reinforcement learning (DRL) to the specific applications and challenges addressed in this thesis.

8.2.1 The Evolution of DRL: Concepts and Foundations

Chapter 1 laid the groundwork by exploring the fundamental principles of DRL. It discussed the evolution of reinforcement learning (RL) from its theoretical origins to its integration with deep learning techniques, providing the conceptual framework that underpins the rest of the thesis.

8.2.2 State-of-the-Art and Technological Advancements

In Chapter 2, we delved into the state-of-the-art in DRL, highlighting recent technological advancements and their implications for both research and practical applications. This chapter emphasized the rapid growth and potential of DRL in various domains.

8.2.3 Challenges and Opportunities in DRL

Chapter 3 offered an in-depth analysis of the challenges and opportunities in DRL. It critically examined issues such as data efficiency, training complexity, and the transferability of learned policies to real-world scenarios.

8.2.4 Innovations in Data Efficiency and Model-Based RL

Our exploration in Chapter 4 introduced novel methodologies to enhance data efficiency in model-based RL. The multi-teacher knowledge distillation framework proposed here marked a significant step towards more efficient learning algorithms.

8.2.5 Training Complexity and Curriculum Learning

Chapter 5 addressed the training complexity in DRL through the lens of curriculum learning. The development of an automatic curriculum learning framework exemplified a structured approach to training in complex environments.

8.2.6 Model Generalization and Memory-Augmented Neural Networks

In Chapter 6, we tackled the critical issue of model generalization in DRL. The introduction of an internal working memory module opened new doors for the application of memory-augmented neural networks, enhancing the adaptability of DRL models.

8.2.7 Safety Concerns in Network Load Balancing

Chapter 7 brought to the fore the safety concerns in network load balancing. The Hierarchical Policy Learning framework proposed in this chapter represented a novel

approach to mitigating conflicts and enhancing system safety in load-balancing tasks.

8.3 Future Work

My future research will continue to delve deeper into the topics discussed above while also exploring new research directions that aim to bridge the gaps between RL algorithms and their real-world applications. As indicated in [91], a general path to Artificial General Intelligence (AGI) can be accomplished through an RL-based decision-making paradigm. To work towards the goal of AGI and broaden the scope of applied-RL, I intend to explore a range of research topics:

Embodied LLMs. Embodied AI seeks to create intelligent agents that interact with their environment via physical embodiment. The recent success of LLMs has inspired researchers to explore the potential of building embodied LLMs capable of handling various real-world tasks. To realize this objective, there are two paths I want to pursue:

1. *LLMs for Embodied Agents.* Recent research indicates that pre-trained LLMs, such as GPT-4 [14], LLAMA [16], or Flamingo [92], can generate text outputs of planning trajectories [93]. However, translating these language outputs into real agent actions remains a challenge. I plan to address this by exploring the possibility of extending LLMs to a hierarchical structure, seen as an options framework [47]. The higher level comprises text planning instructions, and the lower level generates agent actions

through fine-tuned transformers (e.g., Decision Transformers), where an action sequence is generated conditional on the high-level planning instructions.

2. *Embodied Agents for LLMs.* I am also interested in investigating how to utilize current embodied agents to fine-tune LLMs. At present, LLMs are primarily equipped to handle language-related tasks, and they lack a comprehensive understanding of the real world. Embodied agents interacting with the real world can gather a plethora of trajectories that could be beneficial for LLMs. Therefore, incorporating embodied agents into the training or fine-tuning of LLMs is an urgent question. To this end, I aim to expand current multi-modal LLMs to include actions and rewards, allowing the fine-tuned LLMs to achieve better embodied reasoning capability and reducing their hallucination outputs due to a lack of real-world understanding.

Explainability of RL. The 'black box' nature of many RL algorithms poses a substantial obstacle to their integration into various real-world scenarios. This issue is especially pertinent in safety-critical applications, such as autonomous driving or multi-agent taxi scheduling, etc. Providing clear explanations for an agent's specific actions can aid both users and researchers in understanding the algorithm and acting accordingly. In future research, I plan to enhance the explainability of RL algorithms by developing new techniques to visualize the decision-making process (like a binary tree) or elucidating the learned policy. Recent advancements in explainable AI (XAI) offer promising methodologies for opening the 'black boxes' of deep RL, ranging from interpretable

symbolic decision trees to numerical methods like Shapley Values. Deep RL, which employs a Markov Decision Process for training, can produce innovative solutions but may also contain biases or non-obvious decision paths, making XAI crucial for ensuring safe, bias-free, and understandable solutions.

Ethical issues in RL. Ethical considerations always warrant careful research, particularly in instances where human interaction is involved. In future research, I intend to give more consideration to the following ethical issues when proposing new algorithms:

1. *Bias and Fairness:* RL models, learned from either data or specific environment interactions, unavoidably incorporate and amplify biases and unfair decisions established during the training phase. For example, if an RL agent is used in a financial lending system and trained on historical data where certain demographic groups were unfairly denied loans, the agent might perpetuate this unfair practice.

2. *Privacy:* Training RL agents can potentially infringe on privacy rights. If not properly designed, this privacy data could be exposed to the public, leading to disastrous consequences. For instance, if an autonomous driving RL agent is trained on drivers' data, it could potentially learn and leak sensitive information about the inhabitants.

3. *Misuse of Technology:* Like any technology, RL can be misused for detrimental purposes. Preventing the misuse of RL algorithms remains a significant topic that

warrants further attention.

The vulnerability of RL to security and privacy attacks in applications like healthcare and autonomous driving necessitates a focus on developing robust solutions to these challenges. Methods for defending against data poisoning and adversarial perturbations are of particular interest, as well as the protection of privacy-sensitive data used in RL training.

LLM-based RL agents. The recent success of large language models (LLMs) has shined optimism for the advancement of intelligent agents, with the community making notable strides [94]–[96]. LLMs utilize internet-scale textual data, yet they exhibit profound skills in knowledge acquisition, instruction comprehension, planning, reasoning, and natural language interaction. Recognized as catalysts for Artificial General Intelligence (AGI), LLMs, when transformed into agents with broader perceptual and action capacities, could ascend to the third and fourth world scope levels. These enhanced LLM agents, capable of complex task management through cooperation or competition contributing to a society where humans and AI agents coexist and collaborate.

A significant challenge with current large language models (LLMs) is the issue of hallucination, where they generate incorrect responses or decisions. This limitation might be mitigated by integrating a world model to bolster the planning capabilities of reinforcement learning (RL) agents. Such models are particularly vital for managing long-horizon tasks, including multi-round scenarios. My objective is to develop a world model component that learns system dynamics, like state transitions and reward functions,

from offline datasets. This world model could adopt the form of either a multi-layer perceptron or a transformer model, providing a solid basis for enhancing decision-making and planning in LLM-based RL agents.

Bibliography

[1] J. Kang, J. Wang, C. Hu, X. Liu, and G. Dudek, "A generalized load balancing policy with multi-teacher reinforcement learning," in *GLOBECOM*, IEEE, 2022, pp. 3096–3101.

[2] J. Kang, M. Liu, A. Gupta, C. Pal, X. (Liu, and J. Fu, "Learning multi-objective curricula for robotic policy learning," in *CoRL*, ser. Proceedings of Machine Learning Research, vol. 205, PMLR, 2022, pp. 847–858.

[3] J. Kang, R. Laroche, X. Yuan, A. Trischler, X. Liu, and J. Fu, "Think before you act: Decision transformers with internal working memory," *CoRR*, vol. abs/2305.16338, 2023.

[4] J. Kang, X. Chen, D. Wu, *et al.*, "Hierarchical policy learning for hybrid communication load balancing," in *ICC*, IEEE, 2021, pp. 1–6.

[5] V. Mnih, K. Kavukcuoglu, and e. David Silver, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[6] D. Silver, A. Huang, C. J. Maddison, and e. Arthur Guez, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] A. Filos, P. Tigkas, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal, "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?" In *International Conference on Machine Learning*, PMLR, 2020, pp. 3145–3153.

[9] A. Feriani, D. Wu, Y. T. Xu, *et al.*, "Multi-objective load balancing for multi-band downlink cellular networks: A meta-reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, 2022.

[10] I. Akkaya, M. Andrychowicz, M. Chociej, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[11] D. Wu, J. Kang, Y. T. Xu, *et al.*, "Load balancing for communication networks via data-efficient deep reinforcement learning," in *GLOBECOM*, IEEE, 2021, pp. 1–7.

[12] M. Hessel, J. Modayil, H. van Hasselt, *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI*, AAAI Press, 2018, pp. 3215–3222.

[13] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[14] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, OpenReview.net, 2021.

[16] H. Touvron, T. Lavril, G. Izacard, *et al.*, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023.

[17] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[18] A. Clark, D. de Las Casas, A. Guy, *et al.*, "Unified scaling laws for routed language models," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 4057–4086.

[19] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro, "The role of over-parametrization in generalization of neural networks," in *ICLR (Poster)*, OpenReview.net, 2019.

[20] L. Chen, K. Lu, A. Rajeswaran, *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," in *NeurIPS*, 2021, pp. 15 084–15 097.

[21] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Advances in Neural Information Processing Systems*, 2021.

[22] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[23] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *NIPS*, 2014, pp. 2204–2212.

[24] S. M. A. Eslami, N. Heess, T. Weber, *et al.*, "Attend, infer, repeat: Fast scene understanding with generative models," in *NIPS*, 2016, pp. 3225–3233.

[25] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015. arXiv: `1503.02531`. [Online]. Available: `http://arxiv.org/abs/1503.02531`.

[26] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: A short survey," *arXiv preprint arXiv:2003.04664*, 2020.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[29] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[30] K. Narasimhan, T. D. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," in *EMNLP*, The Association for Computational Linguistics, 2015, pp. 1–11.

[31] O. Gottesman, F. Johansson, M. Komorowski, *et al.*, "Guidelines for reinforcement learning in healthcare," *Nature medicine*, vol. 25, no. 1, pp. 16–18, 2019.

[32] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[33] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[34] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI*, AAAI Press, 2016, pp. 2094–2100.

[35] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *ICLR (Poster)*, 2016.

[36] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48, JMLR.org, 2016, pp. 1995–2003.

[37] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Fall Symposia*, AAAI Press, 2015, pp. 29–37.

[38] M. Fortunato, M. G. Azar, B. Piot, *et al.*, "Noisy networks for exploration," in *ICLR (Poster)*, OpenReview.net, 2018.

[39] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[40] J. Schrittwieser, I. Antonoglou, T. Hubert, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *CoRR*, vol. abs/1911.08265, 2019. arXiv: 1911.08265. [Online]. Available: http://arxiv.org/abs/1911.08265.

[41] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *ICRA*, IEEE, 2018, pp. 7559–7566.

[42] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *NeurIPS*, 2018, pp. 4759–4770.

[43] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[44] E. Camacho, C. Bordons, E. Camacho, and C. Bordons, "Model predictive control and hybrid systems," *Model Predictive control*, pp. 289–310, 2007.

[45] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[46] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[47] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[48] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[49] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint arXiv:1712.00948*, 2017.

[50] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[51] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 27 042–27 059.

[52] M. Xu, Y. Lu, Y. Shen, S. Zhang, D. Zhao, and C. Gan, "Hyper-decision transformer for efficient online policy adaptation," *CoRR*, vol. abs/2304.08487, 2023.

[53] M. Xu, Y. Shen, S. Zhang, *et al.*, "Prompting decision transformer for few-shot policy generalization," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 24 631–24 645.

[54] K. Lee, O. Nachum, M. Yang, *et al.*, "Multi-game decision transformers," in *NeurIPS*, 2022.

[55] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[56] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *CVPR*, IEEE Computer Society, 2017, pp. 7130–7138.

[57] S. Lee and B. C. Song, "Graph-based knowledge distillation by multi-head attention network," in *BMVC*, BMVA Press, 2019, p. 141.

[58] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.

[59] S. You, C. Xu, C. Xu, and D. Tao, "Learning from multiple teacher networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1285–1294.

[60] F. Yuan, L. Shou, J. Pei, *et al.*, "Reinforced Multi-Teacher Selection for Knowledge Distillation," in *AAAI*, AAAI Press, 2021, pp. 14 284–14 291.

[61] S. Das, C. L. Giles, and G.-Z. Sun, "Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory," in *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, vol. 14, 1992.

[62] J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," *Neural Computation*, vol. 4, no. 1, pp. 131–139, 1992.

[63] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[64] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, PMLR, 2016, pp. 1842–1850.

[65] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," *Advances in neural information processing systems*, vol. 29, 2016.

[66] T. Munkhdalai and H. Yu, "Meta networks," in *International conference on machine learning*, PMLR, 2017, pp. 2554–2563.

[67] R. Csordás and J. Schmidhuber, "Improving differentiable neural computers through memory masking, de-allocation, and link distribution sharpness control," *arXiv preprint arXiv:1904.10278*, 2019.

[68] H. Ramsauer, B. Schäfl, J. Lehner, *et al.*, "Hopfield networks is all you need," *arXiv preprint arXiv:2008.02217*, 2020.

[69] C.-Y. Wu, Y. Li, K. Mangalam, *et al.*, "Memvit: Memory-augmented multiscale vision transformer for efficient long-term video recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 587–13 597.

[70] S. Sukhbaatar, J. Weston, R. Fergus, *et al.*, "End-to-end memory networks," *Advances in neural information processing systems*, vol. 28, 2015.

[71] T. Munkhdalai, A. Sordoni, T. Wang, and A. Trischler, "Metalearned neural memory," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[72] A. Goyal, A. R. Didolkar, A. Lamb, *et al.*, "Coordination among neural modules through a shared global workspace," in *ICLR*, OpenReview.net, 2022.

[73] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 382, ACM, 2009, pp. 41–48.

[74] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: A short survey," *arXiv preprint arXiv:2003.04664*, 2020.

[75] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 6292–6299.

[76] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," *arXiv preprint arXiv:1707.05300*, 2017.

[77] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *international conference on machine learning*, PMLR, 2017, pp. 1311–1320.

[78] R. Kwan, R. Arnott, R. Paterson, R. Trivisonno, and M. Kubota, "On mobility load balancing for LTE systems," in *Proceedings of the 72nd IEEE Vehicular Technology Conference, VTC Fall 2010, 6-9 September 2010, Ottawa, Canada*, IEEE, 2010, pp. 1–5. DOI: 10 . 1109 / VETECF . 2010 . 5594565. [Online]. Available: https://doi.org/10.1109/VETECF.2010.5594565.

[79] Y. Yang, P. Li, X. Chen, and W. Wang, "A high-efficient algorithm of mobile load balancing in LTE system," in *Proceedings of the 76th IEEE Vehicular Technology Conference, VTC Fall 2012, Quebec City, QC, Canada, September 3-6, 2012*, IEEE, 2012, pp. 1–5. DOI: 10 . 1109 / VTCFall . 2012 . 6398873. [Online]. Available: https://doi.org/10.1109/VTCFall.2012.6398873.

[80] S. S. Mwanje and A. Mitschele-Thiel, "A q-learning strategy for LTE mobility load balancing," in *24th IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications, PIMRC 2013, London, United Kingdom, September 8-11, 2013*, IEEE, 2013, pp. 2154–2158. DOI: 10.1109/PIMRC.2013.6666500. [Online]. Available: https://doi.org/10.1109/PIMRC.2013.6666500.

[81] D. Tennakoon, S. Karunarathna, and B. Udugama, "Q-learning approach for load-balancing in software defined networks," in *2018 Moratuwa engineering research conference (MERCon)*, IEEE, 2018, pp. 1–6.

[82] S. S. Mwanje, L. Schmelz, and A. Mitschele-Thiel, "Cognitive cellular networks: A q-learning framework for self-organizing networks," *IEEE Trans. Network and Service Management*, vol. 13, no. 1, pp. 85–98, 2016. DOI: 10.1109/TNSM.2016.2522080. [Online]. Available: https://doi.org/10.1109/TNSM.2016.2522080.

[83] P. M. Luengo, R. Barco, J. M. Ruiz-Avilés, I. de la Bandera, and A. Aguilar, "Fuzzy rule-based reinforcement learning for load balancing techniques in enterprise LTE femtocells," *IEEE Trans. Veh. Technol.*, vol. 62, no. 5, pp. 1962–1973, 2013. DOI: 10.1109/TVT.2012.2234156. [Online]. Available: https://doi.org/10.1109/TVT.2012.2234156.

[84] X. Chen, L. Kong, X. Liu, L. Rao, F. Bai, and Q. Xiang, "How cars talk louder, clearer and fairer: Optimizing the communication performance of connected vehicles via online synchronous control," in *INFOCOM*, IEEE, 2016, pp. 1–9.

[85] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Deep reinforcement learning based mobility load balancing under multiple behavior policies," in *ICC*, IEEE, 2019, pp. 1–6.

[86] P. H. Barros, I. Cardoso-Pereira, L. Foschini, A. Corradi, and H. S. Ramos, "Load balancing in D2D networks using reinforcement learning," in *2019 IEEE Symposium*

on Computers and Communications, ISCC 2019, Barcelona, Spain, June 29 - July 3, 2019, IEEE, 2019, pp. 1–6. DOI: 10.1109/ISCC47284.2019.8969767. [Online]. Available: https://doi.org/10.1109/ISCC47284.2019.8969767.

[87] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-uav-enabled load-balance mobile-edge computing for iot networks," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6908, 2020.

[88] O. Ahmed, F. Träuble, A. Goyal, *et al.*, "Causalworld: A robotic manipulation benchmark for causal structure and transfer learning," in *ICLR*, OpenReview.net, 2021.

[89] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[90] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 104–114.

[91] Y. LeCun, "A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27," *OpenReview.net*, 2022.

[92] J.-B. Alayrac, J. Donahue, P. Luc, *et al.*, "Flamingo: A visual language model for few-shot learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23 716–23 736, 2022.

[93] W. Huang, F. Xia, T. Xiao, *et al.*, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[94] I. Gur, H. Furuta, A. Huang, *et al.*, "A real-world webagent with planning, long context understanding, and program synthesis," *arXiv preprint arXiv:2307.12856*, 2023.

[95] X. Deng, Y. Gu, B. Zheng, *et al.*, "Mind2web: Towards a generalist agent for the web," *arXiv preprint arXiv:2306.06070*, 2023.

[96] H. Furuta, O. Nachum, K.-H. Lee, Y. Matsuo, S. S. Gu, and I. Gur, "Multimodal web navigation with instruction-finetuned foundation models," *arXiv preprint arXiv:2305.11854*, 2023.