UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Deep Reinforcement Learning Conditioned on the Natural Language

by

Yunqiu Xu

Thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy
under the supervision of Ling Chen and Chengqi Zhang

Australian Artificial Intelligence Institute
Faculty of Engineering and Information Technology
University of Technology Sydney

May 2022

Certification of Original Authorship

I, Yunqiu Xu, declare that this thesis is submitted in fulfilment of the requirements for the award of degree of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

Signed:

Production Note:
Signature removed prior to publication.

Date: 13/01/2022

Abstract

Language-conditional reinforcement learning refers to the reinforcement learning task where the language information serves as essential components in the problem formulation. In recent years, the advances of deep reinforcement learning and language representation learning lead to increasing research interest in this cross-domain topic, which brings benefits to the studies in both language learning and reinforcement learning. However, challenges arise along with the premises, hindering the language-conditional reinforcement learning from being applied in the real world. In this research, we aim at designing language-conditional RL agent that is capable of handling the major challenges.

We first address the challenges in state representation learning under partial observability. Motivated by the premises of the transformer architecture in natural language processing, we design an adaptable transformer-based state representation generator featured with reordered layer normalization, weight sharing and block-wise aggregation. We empirically validate our method on both synthetic and man-made text-based games with different settings. The proposed method show higher sample efficiency in solving single synthetic games, better generalizability in solving unseen synthetic games, and better performance in solving complex man-made games.

Secondly, we study the reasoning process in language-conditional reinforcement learning. The reasoning ability enables the agent to generate the actions with the support of an explainable inference procedure. To achieve this ability, we propose an agent featured with the stacked hierarchical attention mechanism. Through exploiting the structure of the knowledge graph, this agent is able to explicitly model the reasoning process. Our agent demonstrates effectiveness on a range of man-made text-based games.

Thirdly, we study the generalization problem in language-conditional RL. We consider the knowledge graph-based observation, and address this challenge by designing a two-level hierarchical RL agent. In the high level, we use a meta-policy for task decomposition and subtask selection. Then, in the low level, we use a sub-policy for subtask-conditioned action selection. In a series of 8

game sets with different generalization types and game difficulty levels, our proposed agent enjoys generalizability and yields favorable performance.

Finally, we provide solutions to the challenges of low sample efficiency and large action space. We introduce the world-perceiving modules, which automatically decompose tasks and prune actions by answering questions about the environment. We then propose a two-phase training framework to decouple language learning from reinforcement learning, which further improves the sample efficiency. We empirically demonstrate that the proposed method not only achieves improved performance with high sample efficiency, but also exhibits robustness against compound error and limited pre-training data.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors, Prof. Ling Chen and Prof. Chengqi Zhang for their continuous and unconditional assistance throughout the journey in pursuing this degree. They supported me at every stage of my PhD studies, such as leading me to the area of language-conditional reinforcement learning, encouraging me to develop not only the in-depth knowledge in my research topic but also broad research interests in machine learning, helping me to establish connections with academic and industrial bodies, and providing me with valuable career & life advice. Without your patience, encouragement and persistent help, I will never grow into a professional researcher.

Besides my supervisors, I would like to offer my special thanks to Dr. Meng Fang, for providing me with hand in hand guidance to deliver high quality research. Thank you for aiding me to formulate a project management-like research schedule. I benefit from the close and inspiring discussions, which cover almost every aspect of details about how to conduct a research project, such as the problem formulation, model development, experiment design, and the manuscript preparation. Without your help, I probably waste a lot of time on detours.

I would like to acknowledge Dr. Yali Du, Dr. Gangyan Xu, Dr. Joey Tianyi Zhou, Dr. Yang Wang, Dr. Binbin Huang, for providing helpful suggestions throughout our collaboration. I would also like to mention those I met at UTS, specifically Dr. Wei Wu, Dr. Hong Yang, Dr. Yaqiong Li, Dr. Jiamiao Wang, Dr. Jun Li, Mr. Yu Liu, Mr. Shaoshen Wang, Mr. Yayong Li, Ms. Yang Zhang, and Mr. Zihan Zhang. It's my pleasure to work with all of you. In particular, I would like to thank Prof. Yi Yang, for introducing me to Prof. Ling Chen.

I am proud of my parents and grandparents. Thank you for raising me up, providing me with a joyful childhood, and inspiring my curiosity in science and engineering. Thank you for expressing your endless love and support across the ocean. This thesis is dedicated to my grandfather, a veteran passed away in the last year of my PhD study. Thank you, for fostering me a rigorous attitude, and teaching me to be tough to survive hardships.

Last but not the least, I would like to thank my wife, Ms. Yang Liu, for accompanying me through thick and thin.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivations

The sequential decision making is a core topic of machine learning and artificial intelligence. It is related to a situation where the current decision can have both immediate and long-term effects. For many environments, the dynamics is inaccessible and the outputs, such as the labels, are not well-defined. To achieve the final goal (e.g., solving a game, or completing a manipulating task), the decision maker (agent) has to learn from the experience, which is collected through trial-and-error. The Reinforcement Learning (RL) [190], whose goal is to map the environment states to appropriate actions to maximize the feedback signals, is a principal framework for such problems. The traditional RL methods are limited within tabular cases, where both the action space and state space are small and of low dimension. In the past few years, the deep learning techniques help RL to build state representation from high dimensional inputs, and perform stable function approximation over a large state and action space. The resulted method, known as the Deep Reinforcement Learning (DRL), has proven its effectiveness in a large set of challenging tasks including game playing [144, 145, 182], robotics control [68, 111, 115] and many other real world applications [125].

One of the most crucial factors in human evolution is the development of language. Through using language to control the complex social coordination, humans have formulated a modern society that has achieved domination over other life on earth [155, 168]. The abilities of learning

and understanding the language, which are fundamental to humans, are essential in pursuing the human-level AI. The related research topics, such as the Natural Language Understanding (NLU) and Natural Language Processing (NLP) techniques, have been studied for decades [4, 24, 136]. While the past research in language learning focused more on supervised learning and unsupervised learning, recent years have witnessed increasing research interests in integrating language learning with reinforcement learning. Apart from those adopting RL in NLP tasks [38, 124, 161, 170], the language-conditional RL attracts a lot of attentions in the RL research community [133]. The language-conditional RL aims at grounding natural language into reinforcement learning, that it follows RL's problem formulation with the language information serves as unavoidable components. For example, in the text-based games [53, 196], both the observation space and the action space are in the context of texts, that the agent observes textual descriptions and generates textual commands as the actions. Another example is the instruction following tasks, where the language serves as the instructions or subgoals to guide the agent towards the final goal [44, 142, 205].

The earlier attempts of language-conditional RL have been limited to synthetic language [103], small vocabulary space [90], and tabular-wise RL algorithms [34, 35]. The advances in deep reinforcement learning [84, 143, 145] and language representation learning [37, 59, 164] techniques make it possible to scale up the task towards more difficult scenarios, which are closer to the real world applications. Most of the existing works have focused on instruction following and reward modeling, while the tasks with language-based observation space and action space, are far less studied, which is partly due to the lack of suitable platforms. In light of the video game platforms [27, 106], which significantly boost the development of visual-based RL, some text-based games platforms have been recently proposed to facilitate the agent design and performance measurement [53, 85, 196], making it worthwhile to revisit this branch of studies.

As a cross-domain research topic, studying language-conditional RL brings benefits to both NLP and RL. From the perspective of NLP, it enables language learning in an interactive way, which has been proven to be more effective than learning with static data [23, 66, 121]. From the perspective of RL, incorporating language learning helps to address some challenges in reinforcement learning, thus promoting RL's application in the real life. For example, existing works have leveraged the compositional nature of language to improve RL agent's generalizability to unseen instructions [52, 105].

1.2 Research Challenges

Along with the premises, new challenges have appeared in studying language-conditional RL [63, 85]. The research challenges include, but are not limited to learning effective presentations from the language, partial observability, commonsense reasoning, generalization to unseen scenarios, long-term credit assignment with sparse reward, exploration v.s., exploitation trade-off, combinatorial action space, and low sample efficiency. Some of these challenges inherit from either language learning (e.g., language representation learning) or reinforcement learning (e.g., low sample efficiency), while some challenges arise from combining these two areas (e.g., combinatorial action space). These challenges are not strictly exclusive, that addressing one challenge will be helpful in handling other challenges. For example, studying language representation learning not only helps to construct effective state representations, but also helps to alleviate partial observability [8]. Our goal for this research is to design language-conditional RL agent that is capable of handling the major challenges summarized as follows:

1. Language-based state representation learning: While the convolutional neural networks and the multilayer perceptrons are commonly used for processing the visual-based observations, how to build good state representations upon the language-based observations?

2. Partial observability: How to design the state representation learning process as well as the algorithmic optimizing process to address the partially observable settings, where the state can not be directly observed from the environment?

3. Reasoning: How to empower RL agents with the reasoning abilities to extract supporting facts from the language-based inputs, thus yielding better performance and interpretability?

4. Generalization: The RL agents tend to overfit the training environment, how to improve their generalizability towards unseen environments?

5. Combinatorial action space: Different from the video games, where there typically exists a small set of actions, the language-conditional RL settings may involve a large, combinatorial action space. Under such condition, how to reduce the action space to prevent the agent from attempting irrelevant or inferior actions?

6. Low sample efficiency: The low sample efficiency is a crucial limitation of RL, that a huge amount of interaction data is required for training an RL agent. How to improve the sample efficiency to promote RL's practical usage?

1.3 Contributions

This thesis studies the language-conditional reinforcement learning. We pay more attention to the settings where both the observation space and the action space are language-based, while other settings, such as instruction following and language-based reward modeling, will also be involved. We leverage the recent proposed text-based games platforms as our test-beds to analyse and design RL agents, thus providing solutions to the above-mentioned challenges. First, this thesis studies state representation learning for language-conditional RL, that we design an adaptable transformer-based state representation generator featured with reordered layer normalization, weight sharing and block-wise aggregation. Second, this thesis studies RL reasoning, and proposes an agent featured with the stacked hierarchical attention mechanism to achieve the reasoning ability. Third, this thesis addresses the generalization problem by designing a two-level hierarchical RL agent. Finally, this thesis addresses the challenges of low sample efficiency and large action space by introducing the world-perceiving modules, which are capable of automatic task decomposition and action pruning through answering questions about the environment.

The contributions of our research are summarized as follows:

- **Transformer-based State Representation Generator:** This work is one of the first to study the transformer architecture for language-conditional RL tasks. In this work, we show the effectiveness of the transformer architecture in building state representations from the textual observations. We propose a simple transformer-based state representation generator, which could be treated as a plugin for different agents & frameworks. We empirically validate our model in text-based games with different settings, and show that our model outperforms both the vanilla transformer and strong baselines.
- **Stacked Hierarchical Attention Mechanism:** This work is the first step to explore the reasoning ability in language-conditional reinforcement learning. In this work, we introduce

the reasoning process in decision making through considering a knowledge graph as multiple sub-graphs. We design an RL agent, which is equipped with the stacked hierarchical attention mechanism for reasoning. We empirically validate the effectiveness of our agent in a set of man-made games. Our proposed agent achieves favorable performance in comparison with the state-of-the-art baselines.

- **Hierarchical Knowledge Graph Agent:** This work is the one of the first to address the generalization problem in language-conditional reinforcement learning from the perspective of hierarchical RL. In this work, we propose a hierarchical agent upon the knowledge graph-based observation for adaptive subtask decomposition and subtask selection. We show that our agent improves generalizability in solving multiple sets of games with various difficulty levels.

- **Question-guided World-perceiving Agent:** This work addresses the challenges in low sample efficiency and large action space in language-conditional RL. We introduce the world-perceiving modules, which are capable of automatic task decomposition and action pruning through answering questions about the environment. We then propose a two-phase training framework to decouple language learning from reinforcement learning to further improve training efficiency with limited data. We empirically validate the effectiveness and robustness of the proposed method in complex games.

1.4 Thesis Outline

Table 1.1 displays the structure of this thesis. The detailed research roadmap is organized as follows:

Chapter 2: This chapter provides comprehensive literature review about the reinforcement learning as well as language-conditional reinforcement learning.

Chapter 3: This chapter provides preliminaries about the text-based games, which are our test-beds in this thesis

Chapter 4: This chapter provides solution to Challenge 1 and Challenge 2. In order to build effective state representation, thus facilitating decision making and alleviating the problem of partial

TABLE 1.1: Thesis structure

Chapters	Research Tasks
2	Literature Review
3	Preliminaries
4	Transformer-based State Representation Generator
5	Stacked Hierarchical Attention Mechanism
6	Hierarchical Knowledge Graph Agent
7	Question-guided World-perceiving Agent
8	Conclusions and Future Work

observability, we design an adaptable transformer-based state representation generator featured with reordered layer normalization, weight sharing and block-wise aggregation.

Chapter 5: This chapter studies Challenge 3. The reasoning ability enables the agent to generate the actions with the support of an explainable inference procedure. To achieve this ability, we propose an agent featured with the stacked hierarchical attention mechanism. Through exploiting the structure of the knowledge graph, this agent is able to explicitly model the reasoning process.

Chapter 6: This chapter investigates Challenge 4. We consider the knowledge graph-based observation, and address this challenge by designing a two-level hierarchical RL agent. In the high level, we use a meta-policy for task decomposition and subtask selection. Then, in the low level, we use a sub-policy for subtask-conditioned action selection.

Chapter 7: This chapter addresses Challenge 5 and Challenge 6. We introduce world-perceiving modules to automatically decompose tasks and prune actions through answering environment-related questions. We then propose a two-phase training framework to decouple language learning from reinforcement learning to further improve the sample efficiency.

Chapter 8: This chapter concludes the thesis, and identifies future directions.

1.5 Publications

The main body of this thesis has been published in (or submitted to) major artificial intelligence conferences:

- Chapter 4 is based on the paper [214] titled as "Deep Reinforcement Learning with Transformers for Text Adventure Games", which has been published in CoG-2020;
- Chapter 5 is based on the paper [216] titled as "Deep Reinforcement Learning with Stacked Hierarchical Attention for Text-based Games", which has been published in NeurIPS-2020;
- Chapter 6 is based on the paper [215] titled as "Generalization in Text-based Games via Hierarchical Reinforcement Learning", which has been published in EMNLP-Findings-2021;
- Chapter 7 is based on the paper titled as "Perceiving the World: Question-guided Reinforcement Learning for Text-based Games", which has been submitted to ACL-2022;

During my PhD studies, I have published 6 research papers, and have 3 manuscripts in submission:

1. **Yunqiu Xu**, Meng Fang, Ling Chen, Yali Du, Joey Tianyi Zhou, and Chengqi Zhang, "Perceiving the World: Question-guided Reinforcement Learning for Text-based Games", **submitted to** *the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. **(CoRE A*)**
2. Dongwon Kelvin Ryu, Ehsan Shareghi, Meng Fang, **Yunqiu Xu**, Shirui Pan and Reza Haf, "Fire Burns, Swords Cut: Commonsense Inductive Bias for Exploration in Text-based Games", **submitted to** *the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022. **(CoRE A*)**
3. Meng Fang, **Yunqiu Xu**, Yali Du, Ling Chen and Chengqi Zhang, "Goal Randomization for Playing Text-based Games without a Reward Function", **submitted to** *the International Conference on Learning Representations (ICLR)*, 2022. **(CoRE A*)**
4. **Yunqiu Xu**, Meng Fang, Ling Chen, Yali Du, and Chengqi Zhang, "Generalization in Text-based Games via Hierarchical Reinforcement Learning", in *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP-Findings)*, 2021. **(CoRE A)**
5. **Yunqiu Xu**, Meng Fang, Ling Chen, Gangyan Xu, Yali Du, and Chengqi Zhang, "Reinforcement Learning With Multiple Relational Attention for Solving Vehicle Routing Problems", in *IEEE Transactions on Cybernetics*, 2021. **(CoRE A)**

6. **Yunqiu Xu**, Meng Fang, Ling Chen, Yali Du, Joey Tianyi Zhou, and Chengqi Zhang, "Deep Reinforcement Learning with Stacked Hierarchical Attention for Text-based Games", in *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 16495-16507, 2020. **(CoRE A*)**

7. **Yunqiu Xu**, Ling Chen, Meng Fang, Yang Wang, and Chengqi Zhang, "Deep Reinforcement Learning with Transformers for Text Adventure Games", in *IEEE Conference on Games (CoG)*, pages 65-72, 2020. **(CoRE C)**

8. Binbin Huang, Zhongjin Li, **Yunqiu Xu**, Linxuan Pan, Shangguang Wang, Haiyang Hu and Victor Chang, "", in *Wireless Communications and Mobile Computing*, 2020. **(CoRE B)**

9. Binbin Huang, Yangyang Li, Zhongjin Li, Linxuan Pan, Shangguang Wang, **Yunqiu Xu** and Haiyang Hu, "Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing", in *Wireless Communications and Mobile Computing*, 2019. **(CoRE B)**

Chapter 2

Literature Review

The literature review is organized as follows. Section 2.1 introduces the background and the basic concepts about reinforcement learning. Section 2.2 focuses on the language-conditional RL, which is the core domain of this thesis. Section 2.3 reviews the RL (especially DRL) techniques from the perspective of real world applications, and identifies the challenges.

2.1 Reinforcement Learning

Reinforcement learning is the area of machine learning where an agent learns to perform sequential decision-making in an environment. Generally, RL is assumed to obey the Markov property, that the future of decision making depends on the current state only. Following this assumption, RL can be formulated as a Markov Decision Process (MDP) [28], which is a 5-tuple consisting of the state set $\mathcal{S}$, the action set $\mathcal{A}$, the transition function $T(s_{t+1}|s_t, a_t)$, the reward function $R(s_t, a_t, s_{t+1})$ and discount factor $\gamma \in (0, 1]$. Fig. 2.1 shows the interaction process. At each time step t, the agent will be presented with a state $s_t \in \mathcal{S}$. By selecting an action to interact with the environment, the state will transit to a new one s_{t+1}, and the agent will receive a reward r_{t+1} as the feedback. The objective of reinforcement learning is the maximization of the future cumulative return R_t, which is the γ-discounted sum of rewards:

$$R_t = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{2.1}$$

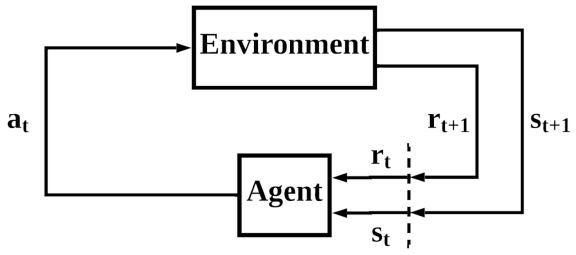


FIGURE 2.1: The interaction process in the RL problem

The RL agent is designed to be with one or more of following components: the policy function, the value function and the environment model. The policy function, which is the agent's behavior function, maps a state to an action $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The value function is the estimation of future return given a state. The environment model is the estimation of environmental components, such as the reward function and the transition dynamics function. The goal of the agent is to learn an optimal policy π^* to maximize the expected return R_t.

According to [70, 190], the RL algorithms can be categorized as different classes. One main classification approach is to divide them as the value-based methods, the policy-based methods, and the hybrid of them. The value-based algorithms estimate the optimal value function, and guide the RL agent towards the state with maximum future value. The policy-based algorithms optimize the policy function directly. The Actor-Critic is a category of algorithms that combine the policy functions with the value functions. By default, the value-based methods and the policy-based methods are model-free. An RL algorithm is regarded as model-based, if the environment model is used to simulate the transition dynamics and perform planning, otherwise model-free. Figure 2.2 displays the schema of different RL methods, where the experience denotes the interaction data for training. There are other classification criteria. Depending on whether the policies for evaluating and decision making are same, these algorithms can be classified as on-policy algorithms, or off-policy algorithms. Another division is the online learning v.s., offline learning, which depends on whether the agent has access to the environment, or just be provided with a static dataset of transitions [123]. The rest of this section reviews the commonly used RL algorithms with the division of value-based, policy-based and model-based. Besides the traditional RL algorithms designed for the tabular cases, this section also demonstrates how the deep learning techniques could be applied in solving more complex scenarios.

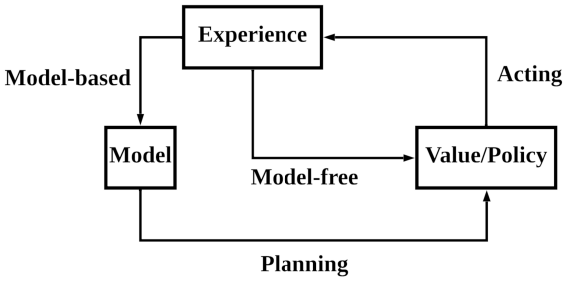


FIGURE 2.2: The schema of different RL algorithms.

2.1.1 Value-based RL

The value-based RL methods aim at estimating the value (e.g. the expected return) of all states accurately. The optimal policy π^* corresponds to the optimal state-value function:

$$V^*(s) = \max_{\pi} \mathbb{E}[R|s, \pi], \forall s \in \mathcal{S} \tag{2.2}$$

The action leading to next state s_{t+1} with largest value will then be selected. Since the agent usually does not have the access to the transition dynamics function, the state-value function is extended to the state-action-value function, i.e., the Q value function $Q^{\pi}(s, a) = \mathbb{E}[R|s, a, \pi]$. One of the most widely used value-based methods is Q learning [208], which follows the Bellman optimality equation to estimate the Q values:

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \mathbb{E}[\max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a')] \tag{2.3}$$

The Q-learning is limited to the tabular cases, since the agent need to traverse all of the state-action pairs to estimate the Q-value. Through deep learning-based representation learning and value function approximation, the Deep Q-network (DQN) [144, 145] extends Q-learning to high-dimensional state space. Specifically, DQN uses the neural networks to extract state representation from visual observations, so that the curse of dimensionality is alleviated. Besides, the value function is parameterized via neural networks, so that there's no need for estimating the Q values of all state-action pairs. In order to handle the instability during function approximation, DQN further considered two techniques, the experience replay and the target network. The experience replay [128] is a memory buffer to store the transitions collected from the environment. During training, a batch of transitions will be sampled from this buffer to optimize the algorithm. In this

way, the temporal correlations between transitions, which may affect learning, are broken. The target network is another way to stabilize learning. It's a copy of the policy network to estimate the Q value for the next state. Different from the policy network, the target network is updated less frequently to handle non-stationary. DQN has achieved human-level performance and surpassed previous methods in playing a large number of Atari 2600 games [27]. Going beyond DQN, some further improvements have been recently proposed, including handling over-estimation via a double estimator [84], decoupling value and advantage function [207], pritorizing transitions [172], estimating the value distribution [54], introducing the noisy parameters to help exploration [69], and the integration of these improvements [91]. While these extensions mainly focus on the Q value estimation, there are also works enhancing the value-based framework by introducing hierarchy [120, 199], auxiliary rewards [26, 158], memory modules [87], and action elimination techniques [62, 227].

2.1.2 Policy-based RL

Compared to the value-based RL, which maintains a value function, the policy-based RL directly searches for the optimal policy. The search strategies can be either gradient-free or gradient-based. While the gradient-free strategies, such as the evolutionary algorithms [171], require running large populations of agents in parallel, most DRL-related methods are in favor of the gradient-based strategies. For a policy π parameterized by θ, the objective function can be formulated as:

$$L(\theta) = \mathbb{E}[R|\pi(s_0, \theta)] \tag{2.4}$$

where R is the total discounted return. The policy can be either stochastic or deterministic. A stochastic policy is defined as the probabilities of taking actions given a state:

$$\pi(s, a) = \mathbb{P}[a|s] \tag{2.5}$$

A deterministic policy outputs the action with largest probability:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} \pi(a|s) \tag{2.6}$$

For a stochastic policy, the gradient can be estimated via the REINFORCE method [211]. In this method, the estimation of R is obtained through Monte Carlo sampling over the whole episode, which incurs high variance. One general way for variance reduction is to subtract the return with a baseline, which can be represented as the average episodic return [211], or the value estimated by a critic network [144].

Both the stochastic policy and the deterministic policy can be cast into the Actor-Critic framework [117] that the "Actor" denotes the policy and the "Critic" denotes the state-value estimator. The learning process can be accelerated through estimating the Q values using n-step bootstrapping. If π is deterministic, and the value $Q(s, a)$ is differentiable (e.g., for tasks with the continuous action space), the policy gradient can be estimated via the Deep Deterministic Policy Gradient (DDPG) method [126]. The Asynchronous Advantage Actor-Critic (A3C) framework [143] is another famous extension of Actor-Critic. In A3C, multiple actors are executed in parallel and trained asynchronously, and the value estimated by the critic network also serves as the baseline for variance reduction. Recently, some other practices, such as enhancing DDPG with delayed policy updates [72], and integrating the actor-critic with energy-based regularization [80, 81], have been proposed and achieved promising performance.

To stabilize the policy updating process, the trust region-based methods, are introduced to restrict the updating steps within a region. In this way, the updated policy is prevented from deviating too wildly from the previous policy. For example, the Kullback-Leibler divergence (KLD) can be utilized to measure the difference between the current and proposed policies, thus constraining the update step length [109, 110, 174]. The recent advances include introducing a generalized advantage estimator for variance reduction [175], conducting the optimization using the Kronecker-factored approximate curvature [212], simplifying the second-order approximation of the KL divergence to first-order [176], and constraining the optimization using a constrained MDP [1].

2.1.3 Model-based RL

The value-based and policy-based methods are generally model-free, that the agent learns by directly interacting with the environment. Different from these model-free algorithms, the model-based RL algorithms are equipped with the environment model to simulate the transitions and perform planning, leading to higher sample efficiency. This is particularly important to the real world

deployments, in which the transition data is limited and expensive to collect. Dyna-Q [189] is a classic model-based framework, which combines learning and planning in tabular case. The PILCO algorithm [58] addressed the continuous action space through learning a probabilistic dynamics model with the Gaussian process. The rise of deep learning makes it available to design environment models for high-dimensional scenarios, such as the video games and the robotics control tasks, leading to a wide range of related studies [68, 79, 82, 108, 111, 150, 156, 163, 186]. Typically, these works leverage an encoder to encode the input as latent representation for decision making, and a decoder to decode the representation for training or planning. The encoder can either be trained via unsupervised learning, or self-supervised learning. The model predictive planning can be performed within the latent space, or upon the decoded high-dimensional observations. For the model-based RL methods, how to make the model accurate enough to reflect the environmental dynamics is of vital importance, since the both decision making and planning process are based on the model. Besides the planning-based methods, there are also works utilizing the model predictive error to guide exploration [39, 160].

2.2 Language-conditional RL

The language-conditional RL still follows the problem formulation of reinforcement learning, while one or more components, such as the observation space, the action space, the instruction and the reward function, are represented by natural language. There also exists another non-strictly exclusive branch of works being named as the language-assisted RL, which shares similar settings with the language-conditional RL, except that the language information is not essential in solving the tasks. For example, the language can be used for providing additional environment information (e.g., textual descriptions of the entities) [35, 64, 151], or providing a prior to modularize the structure or representations of an agent [13, 14]. We focus on the language-conditional RL in this research.

2.2.1 Language-based Observation and Action Space

We first review language-conditional RL in the context of language-based observation and action space. A classic type of games, the text-based games, are suitable test-beds for this scenario [154].

TABLE 2.1: The DRL agents for text-based games.

Name	Game	Observation	Action	Algorithm
LSTM-DQN [152]	man-made	textual	parser	DQN
DRRN [88]	man-made	textual	choice	DQN
SSAQN [229]	man-made	textual	parser	DQN
LSTM-DRQN [226]	synthetic	textual	parser	DRQN
AEN [227]	man-made	textual	parser	DQN
TDQN [85]	man-made	textual	template	DQN
KG-DQN [8]	synthetic	KG	parser	DQN
CNN-DQN [220]	man-made	textual	parser	DQN
Trans-v-DRQN [214]	synthetic	textual	parser	DRQN
Trans-v-DRRN [214]	man-made	textual	choice	DQN
SC [102]	both	textual	parser	DRQN
LeDeepChef [3]	synthetic	textual	parser	A2C
KG-A2C [7]	man-made	KG	template	A2C
GoExplore Seq2Seq [135]	synthetic	textual	parser	Go-Explore
SHA-KG [216]	man-made	KG	template	A2C
CALM [219]	man-made	textual	choice	DQN
MPRC-DQN [78]	man-made	textual	template	DRQN
Q*BERT [10]	man-made	KG	template	A2C
GATA [2]	synthetic	KG	choice	DDQN
TWC [147]	synthetic	KG	parser	A2C
H-KGA [215]	synthetic	KG	choice	DDQN
CREST [45]	synthetic	textual	parser	DRQN

In the text-based games, the game player observes textual descriptions of the environment state at each time step, and enter a textual command as the response. Due to the lack of effective methods and computing power, early attempts on the text-based games are limited in tabular RL or non-RL methods [34, 35]. Besides, assumptions have been made to reduce the challenges, such as representing the observation via logic programming language instead of natural language [5, 94]. Since 2016, the Text-Based Adventure AI Competition has been launched to promote the research in this domain [20]. By introducing game playing heuristics, some non-RL competitors, such as the BYU16Agent [73], the Golovin agent [119], the CARL agent [20] and the NAIL agent [86], have achieved competitive performance in solving multiple games. For example, the NAIL agent [86] is designed with pre-defined rules for interacting with the objects, exploring the state space, and building the game map to track the states. Despite being effective as game solvers, these agents are hard to be extended to more practical scenarios, since designing those rules requires a huge amount of expert knowledge [85].

The deep reinforcement learning and language representation learning techniques have recently

been applied to the text-based games. Besides, some game platforms have been proposed, making it easier to design game agents and measure their performance. The Jericho platform [85] is designed in light of the OpenAI-Gym [36], that it provides an RL interactive interface for the man-made games, such as Zork 1 [31]. Being originally designed for the human players, the man-made games have more complex logic, and contain much larger state space and action space. The TextWorld platform [53] provides a sandbox-like game interface, that it supports generating multiple games with customizable difficulties and vocabularies, thus facilitating studying generalization [51] and curriculum learning [29]. Another commonly-used platform is LIGHT [196], which supports interacting with multiple players.

Table 2.1 shows the DRL agents designed for solving the text-based games. We classify the agents by the type of games they are applied on, the forms of the observations and actions, and the RL algorithms for optimization. We regard those initially designed for the human players as man-made games (e.g., those supported by Jericho), and those generated by TextWorld or LIGHT as synthetic games. By default, the observation are textual descriptions [152, 226]. Some work also considered constructing knowledge graph from the raw textual observation, thus organizing the information in a structural way [7, 8, 230]. Regarding the form of actions, the parser-based agents generate actions word by word, leading to a huge combinatorial action space [116]. The choice-based agents circumvent this challenge by assuming the access to a set of admissible actions at each game state [88]. The template-based agents achieve a trade-off between the huge action space and the assumption of admissible action set by introducing the template-based action space, where a template is selected first, then be filled with verbs and / or objects [7, 85]. According to [88], there also exists hypertext-based action space, which is seldom considered in studying the language-conditional RL. Existing methods (including our work discussed in this thesis) have been proposed for addressing challenges such as state representation learning [78, 214, 220], partial observability [8, 234], combinatorial action space [219, 227], reasoning [147, 216, 224], and sparse reward [10, 135]

Besides the text-based games, the RL with language-based observation and / or action space can also be conducted upon other domains, such as the dialogue systems and the Question Answering systems (QA). The dialogue systems have been extensively studied, and the reinforcement learning becomes promising in recent years [47]. Compared with the supervised learning-based methods, where a large training dataset has to be collected and may still lack coverage, RL enables the

agent to learn through interacting with the environment, eliminating the need for labeled data [41]. The QA systems can also be conducted in the context of RL, and the form of observation can be extended to visual-based in the Visual QA [17] and Embodied QA tasks [56].

2.2.2 Language-based Instruction Following and Reward Function

In the instruction following tasks, the agent is presented with a sequence of high-level instructions, and executes actions in response to those instructions. Some typical instruction following tasks include navigation [71, 206] and robotics manipulation [142, 202]. In the language-based RL setting, the instructions are represented by natural language, such as the description of a task [105] (e.g., guidance about how to order a set of balls with different colors), or a specific entity [52] (e.g., "pick up the larger blue cat"). The agent is required to understand the instruction, its connection with the environment state, and the actions. This branch of works emphasizes on generalization towards unseen instructions, such as unseen entities, unseen verbs, and unseen verb-entity combinations [169]. The early attempts simplified the natural language as the formal language, that an instruction is decoupled as a combination of entities and their relation, which will be modeled in the object-level [18, 46]. Recently, the deep learning techniques enable directly embedding the human-generated natural language instructions [44, 76, 103, 142, 180]. The cross-modal learning techniques are also involved in the representation learning process to combine the instruction with the state [166, 205, 239].

The language-based instruction following is always accompanied by reward modeling. If we treat the instruction as the goal, or link it to a goal state, the agent will receive reward conditioned on whether it successfully accomplishes the instruction, or whether it observes the goal state [107]. Modeling the reward function is helpful when the environmental reward is sparse or not well defined [21, 52](e.g., no reward function is provided by the environment). A commonly-used reward modeling method is the Inverse Reinforcement Learning (IRL), which models an "intrinsic" reward function to map an instruction to a goal state, and provide the reward [95, 153]. The reward function could be learnt through a joint generative model of the reward, the action and the instruction given the full demonstrations [71, 134], or be learnt through an adversarial process if only the goal-instruction pairs are provided [21]. Such reward modeling can also be conducted through non-RL methods. For example, the ELLA framework [140] trained two classifiers through

supervised learning, where one classifier checks whether an instruction has been accomplished at current state, and another classifier checks whether this instruction is relevant to solving the task. The reward will then be assigned based on whether a relevant instruction is accomplished.

2.3 DRL's Real World Applications and Challenges

Besides achieving great breakthroughs in game playing, the DRL methods have been adopted to a broad range of domains such as robotics, recommendation systems, natural language processing, computer vision, combinatorial optimization, transportation, finance, healthcare, energy and education area. The usage of DRL techniques can be divided as: 1) directly training a decision making policy (e.g., in robotic grasping and manipulation [115]); 2) optimizing the system where it's hard to collect optimal solutions as labels for supervised learning (e.g., in combinatorial optimization tasks [118]); and 3) generating diverse outputs (e.g., in recommendation systems [238] and dialogue systems [124]). Different from the game playing benchmarks, where the environment as well as the components of MDP has been well-defined, in order to apply deep reinforcement learning techniques in real world, one has to pay more attention to how to appropriately formulate the RL problem. According to [125], the pipeline of building an RL application includes problem formulation, data preparation, feature engineering, representation selection, algorithm selection, experiment in simulated environment and final deployment in real environment.

Despite the promises, the DRL techniques still face challenges when being deployed for real life tasks. One typical challenge is how to transfer the trained agent from the simulated environment to real system. Different from the simulation system, in the real system it may be expensive to collect enough data for training, and there are safety concerns for trial-and-error. Recent studies to tackle this challenge include meta learning [67, 165], model-based RL [68, 111], and learning from demonstrations [92, 167]. Another challenge is the large action space. Different from the Atari games, whose action set only contains 18 discrete actions, in the application areas such as the recommendation systems, the action set could be extremely large and changing overtime. For example, in recommendation system there exist millions of recommendable items. While the new items can not be given discrete action indices beforehand, there are also out-dated ones that should not be recommended. Currently, the solutions for this challenge consist of action embedding that embeds the discrete action set to continuous action space [42, 62, 194], building criteria to eliminate

actions with low ranking scores [8, 227], and formulating the MDP to be with stochastic action sets [33, 43]. Other challenges for DRL's real world deployment include system delays, satisfying environmental constraints, partial observability and non-stationarity, multi-objective reward design and credit assignment, and real-time inference [63].

Chapter 3

Preliminaries

We study the language-conditional RL in this thesis. In particular, we center on the setting where the observation and action space are language-based (Chapter 4, Chapter 5, Chapter 6 and Chapter 7). Under this setting, our research also involves language-based instruction following and reward function (Chapter 6 and Chapter 7). Since the text-based games are consistent with our settings, we use these games as the test-beds, and aim at designing RL agents in solving these games. For simplicity, we denote the proposed agents as text-based game agents. In this chapter, we provide the common problem formulations of the text-based games, which will be used throughout the thesis.

3.1 Text-based Games

Text-based games (could also be called "Text Adventures" or "Interactive Fictions") are simulated systems where the agent observes textual descriptions, and generates textual commands to interact with the environment. The man-made games (originally designed for human players) have different themes, vocabularies and objectives [85]. For example, in the Sword & Sorcery game "Zork: The Undiscovered Underground" [30], the agent is a warrior being trapped in the dungeon, and the objectives consist of getting the treasure, killing the monsters, and escaping from the dungeon. Another example, "PlanetFall" [138], is a Science Fiction game, where the agent is an astronaut surviving from a spaceship crash, and the objectives include getting out of the spaceship before explosion, finding the food and weapons, beating the aliens, and finally escaping from this strange

Observation: Convention Hall. You are in attendance at the annual Grue Convention, this year a rather somber affair due to the "adventurer famine" that has gripped gruedom in this isolated corner of the empire. All around you, grues are standing; some in conversation, some drinking, some even paying attention to the speaker. There is a trash chute in one of the walls, and a reader board hangs nearby. You can see some grues, a hat, some clothes and some chewed-up shoes here
> Action: take glasses
Observation: You take the pair of glasses and place them under your costume. You had better hurry; the steady gurgling is becoming harder and harder for you to maintain. [Your score has just gone up by five points.]
> Action: remove mask
Observation: You take off the fish-mouthed mask.

FIGURE 3.1: The interface example of a man-made game "Zork: The Undiscovered Underground". The observation (black) can be the description of environment status, or the feedback of the previous action. The action (blue) is a textual command.

planet. Apart from the man-made games, the existing platforms make it possible to generate a set of synthetic games, which are with similar vocabulary and customizable difficulties [53, 196].

Fig. 3.1 shows an interface example of a man-made game "Zork: The Undiscovered Underground". The action is a textual command. The observation can be either the description of the environment status, or the feedback of the previously executed action. The text-based games provide a safe and interactive way to study language-conditional reinforcement learning, language grounding, systematic generalization, and dialogue systems. Solving these games requires the agent to deduct the goals and solving strategies from the natural language, and address the before-mentioned challenges.

3.2 Partially Observable Markov Decision Process

Since the agent can not directly observe the true environment states, these games are formulated as a Partially Observable Markov Decision Process (POMDP) [53]. A POMDP can be described by a 7-tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{A}, T, r, \Omega, O, \gamma \rangle$. $\mathcal{S}$ represents the state set, $\mathcal{A}$ represents the action set, $T(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^+$ represents the state transition probabilities, $r(s,a) : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ represents the reward function, Ω represents the observation set, O represents the conditional observation probabilities, and $\gamma \in (0,1]$ represents the discount factor. For each step, the agent receives an observation $o_t \in \Omega$ based on the probability $O(o_t|s_t, a_{t-1})$, and chooses an action a_t from the action space $\mathcal{A}$. Then, based on the probability $T(s_{t+1}|s_t, a_t)$, a new state will be devised by the environment, and a reward r_{t+1} will be returned to the agent. Similar to the MDP, the agent selects the action with the aim of maximizing the expectation of the discounted cumulative rewards: $R_t = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

3.3 Knowledge Graph

Besides the textual observation, we also consider the setting where the observations are represented by Knowledge Graphs (KGs) in Chapter 5, Chapter 6 and Chapter 7. We define a graph G as a combination of two sets (V, E), where V is the set of nodes, and E is the set of edges. Then we define a triple as $\langle Subj,\ Rel,\ Obj \rangle$, meaning that a node $Subj \in V$ has a relation $Rel \in E$ to another node $Obj \in V$. For instance, $\langle Grue,\ In,\ ConventionHall \rangle$ indicates that there is a $Grue$ In the $ConventionHall$. We then formulate the KG as a set of such triples. Depending on different experimental settings, the KGs could either be built from the textual observation, or be directly provided by the environment.

Chapter 4

Transformer-based State Representation Generator

In this chapter, we study the transformer architecture for state representation learning in language-conditional RL. The transformer architecture has proven to be more effective than the recurrent modules in the language processing tasks. However, in generating state representations for the RL tasks, its potential is seldom exploited, and might be hindered by the huge amount of weight parameters. In this work, we design an adaptable transformer-based state representation generator featured with reordered layer normalization, weight sharing and block-wise aggregation. We empirically validate our method on both synthetic and man-made text-based games with different settings. The proposed method show higher sample efficiency in solving single synthetic games, better generalizability in solving unseen synthetic games, and better performance in solving complex man-made games.

4.1 Introduction

For solving the RL especially the DRL tasks, building an effective representation generator is one of the most crucial steps. In the domain of language-conditional RL, such as the text-based games, existing methods have been targeting at language-based state representation learning [152], large action space [227], delayed credit assignment [226], and partial observability [8]. In order

to process the language-based states, the state representation generator is usually implemented through recurrent modules, such as the Long Short-Term Memory (LSTM) [88, 152, 226] and the Gated Recurrent Unit (GRU) [7, 85]. Recently, the transformer architecture, which is featured with the self-attention mechanism [197], has empirically proven to be more effective than the recurrent modules in a wide range of natural language processing tasks[59, 164, 223]. However, such architecture is seldom investigated as the state representation generator in the RL tasks.

In this work, we aim at designing a state representation generator in solving the language-conditional RL tasks. We are motivated by the premises of the transformer architecture in natural language processing, and believe that such architecture could be effective in obtaining language-based state representation for RL. We first propose a generator upon the vanilla transformer (i.e., the original version without further modifications) to investigate whether this module is more powerful than the recurrent modules. Then we consider three modifications on this module to further improve the performance. Firstly, we reorder the layer normalization operation by putting it inside the residual connections[89], providing an identity map between the input and the output. Secondly, for the purpose of reducing the number of weights, we conduct weight sharing among all transformer blocks. Thirdly, we apply the gating mechanism [96, 187] to aggregate each transformer block's input flow and output flow, thus further enhancing the residual connection.

We conduct experiments on both synthetic and man-made text-based games with different settings. In the single game setting, the proposed method achieves higher sample efficiency that it spends fewer training interactions on learning to solve the games. In the multiple unseen games setting, the proposed method achieves improved generalizability in solving games with unseen layouts. Our model also outperforms the baselines in multiple complex man-made games.

Our contributions through this work are summarized as following aspects:

1. Our work is one of the initial studies to exploit the potential of the transformer architecture for language-conditional RL tasks.

2. We show the effectiveness of the transformer architecture in building state representations from the textual observations.

3. We propose a simple transformer-based state representation generator, which could be treated as a plugin for different agents & frameworks.

4. We empirically validate our model in text-based games with different settings. The results indicate that our model surpasses both the vanilla transformer and strong baselines.

4.2 Related Work

The transformer architecture has been proved to outperform the recurrent modules in multiple language processing scenarios, such as language modeling [59], text summarization [131], text generation [112], and machine translation [197]. However, this architecture is much less investigated in the area of reinforcement learning, where the recurrent modules such as LSTM are still adopted dominantly. In the multi-agent reinforcement learning tasks, the self-attention mechanism has been applied for modeling the objects and their relationship [228]. Parisotto et al. [159] replaced the recurrent module with the transformer architecture, and tackled the problem in optimizing the vanilla transformer. However, other than building the state representation, both the transformer architecture and the recurrent modules are used for storing the historical information. Although Zelinka et al. [230] applied the transformer architecture as a part of the state representation generator, neither analysis nor modification is conducted upon the vanilla transformer. Different from those existing works, we emphasize on digging the potential of the transformer architecture to be the representation generator for language-conditional RL tasks. Particularly, we try to give answers to following research questions:

- Whether we can utilize the transformer in building the state representations?
- Whether the vanilla transformer's optimization problem [141] still exists in our tasks?
- How to further improve the transformer's performance in language-conditional RL?

4.3 Methodology

Fig. 4.1 shows our agent, which contains a representation generator for building the representation from s_t, and an action scorer for computing the Q values. We concentrate on the generator module, which is shown in the dashed box. Fig. 4.2 displays our proposed transformer-based representation

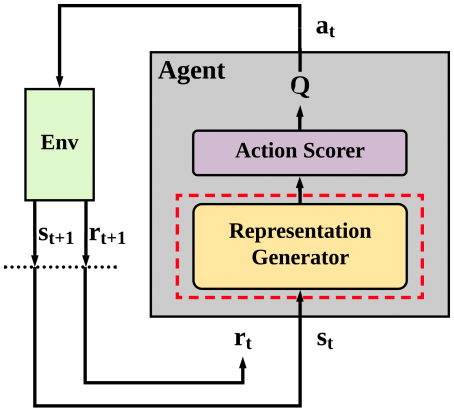


FIGURE 4.1: Our agent, which contains a representation generator for constructing representation from s_t, and an action scorer for computing the Q values. We concentrate on the representation generator, which is shown in the dashed box.

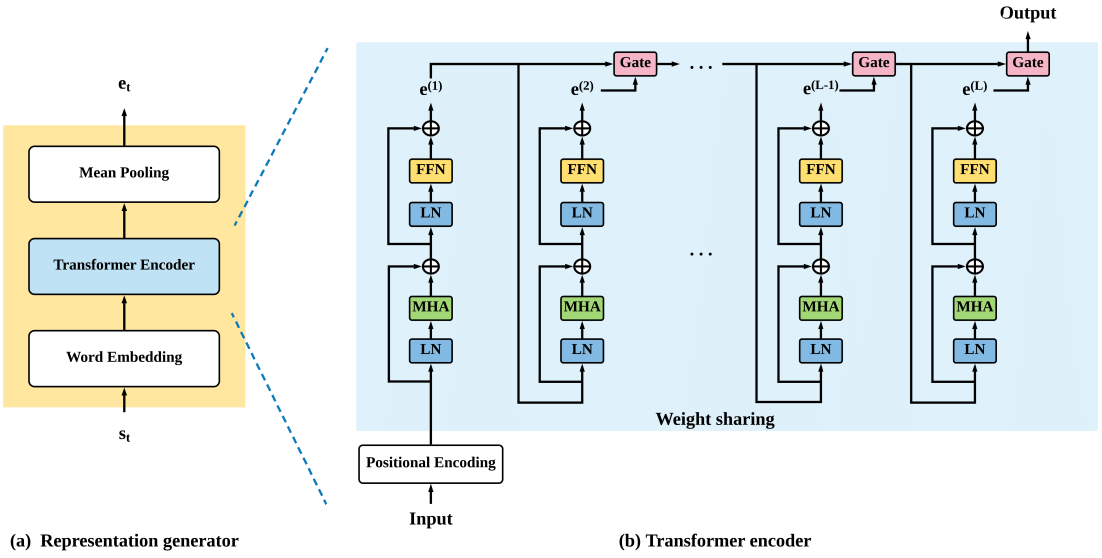


FIGURE 4.2: (a) Our transformer-based representation generator. (b) The transformer encoder, which consists of L transformer blocks. In each transformer block, we reorder the layer normalization operation by putting it inside the residual connections. We share the learnable weights across all transformer blocks (blue region). We then propose gate modules to aggregate the input flow and output flow of each transformer block.

generator, which consists of a word-level embedding layer, the transformer encoder and the mean pooling operation. Built upon the vanilla transformer [197], the transformer encoder consists of L blocks with three enhancements for the language-conditional reinforcement learning. Firstly, we reorder the layer normalization operation by putting it inside the residual connections. Secondly, we conduct weight sharing between the transformer blocks. Thirdly, we apply gating modules to aggregate the input flow and output flow of a transformer block.

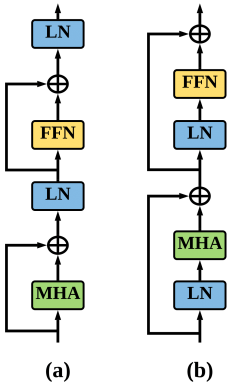


FIGURE 4.3: The vanilla transformer block (a) and the proposed transformer block (b), where the layer normalization operation is put inside each residual connection.

4.3.1 Layer Normalization

Fig. 4.3 compares the blocks for the vanilla transformer (a) and our proposed transformer (b). LN denotes the layer normalization, FFN denotes the feed-forward networks, and MHA denotes the multi-head self-attention. Motivated by the similar operation in the question-answering tasks [223], where an identity map is formed between a sub-module's input and output, we conduct layer normalization reordering by putting LN inside each residual connection. We denote the input embedding of the l-th transformer block ($l \in [1, L]$) as $\boldsymbol{e}^{(l-1)} \in R^{T \times D}$, where T and D denote the sequence length and the embedding dimension, respectively. For the MHA sub-module, the input embedding $\boldsymbol{e}^{(l-1)}$ will be first processed by the layer normalization operation followed by the multi-head attention to obtain the output embedding $\boldsymbol{e}^{(l)}_{\text{MHA}}$:

$$\boldsymbol{e}^{(l)}_{\text{MHA}} = \text{MHA}(\text{LN}(\boldsymbol{e}^{(l-1)})). \tag{4.1}$$

Then $\boldsymbol{e}^{(l-1)}$ and $\boldsymbol{e}^{(l)}_{\text{MHA}}$ will be aggregated by a residual connection:

$$\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}} = \boldsymbol{e}^{(l-1)} + \text{ReLU}(\boldsymbol{e}^{(l)}_{\text{MHA}}). \tag{4.2}$$

For the FFN sub-module, $\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}}$ is treated as the input embedding, and similar operations will be conducted: $\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}}$ will first be processed by the layer normalization operation and some linear layers, then a residual connection is applied to obtain the final output $\boldsymbol{e}^{(l)} \in R^{T \times D}$:

$$\boldsymbol{e}^{(l)} = \widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}} + \text{ReLU}(\text{FFN}(\text{LN}(\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}}))). \tag{4.3}$$

4.3.2 Weight Sharing

Our second modification is to reduce the learnable weights through weight sharing, yielding a lightweight transformer. Typically, the transformer-based models used in language processing tasks (e.g., question answering, machine translation) are less efficient to be trained from scratch, as they have large amounts of learnable weights [59]. Such a problem might be even more noticeable in reinforcement learning, in which the low sample efficiency remains a long-standing challenge [63]. The idea of weight sharing originates from some similar operations in the large-scale language models, for example, the ALBERT [122]. Specifically, as shown in the blue region of Fig. 4.2 (b), the weights are shared among all of the transformer blocks, as well as all of the block-wise gate layers. By doing this, we can make the number of the learnable weights friendly to optimize even if we increase the number of blocks.

4.3.3 Block-wise Gate Layer

Our third modification is to conduct block-wise aggregation after each transformer block, thus enhancing the intra-block residual connections. Similar gating operations can be found in GTrXL [159] with the aim of stabilizing learning. However, while their work conducted gating within the transformer block, we differently apply the gating operations between the blocks. For the l-th transformer block, we aggregate the input flow $\boldsymbol{e}^{(l-1)}$ and output flow $\boldsymbol{e}^{(l)}$ through a gated output connection:

$$\boldsymbol{e}^{(l)} = \boldsymbol{e}^{(l-1)} + \sigma(W_g \boldsymbol{e}^{(l-1)} + b_g) \odot \boldsymbol{e}^{(l)}, \tag{4.4}$$

where W_g and b_g are learnable parameters, and σ denotes the Sigmoid operation.

4.4 Experiments

4.4.1 Experiment Setting

We carry out experiments upon both synthetic and man-made text-based games. The synthetic games are generated based on the TextWorld's CoinCollector Challenge [53]. In each game, the agent is located in a house, and its objective is to explore the rooms to find the unique coin within

TABLE 4.1: The details of the synthetic games in the CoinCollector domain.

Attribute	**L30E**	**L30H**	**L40E**
Rooms	30	90	40
Quest length	30	30	40
Task name	**L30E**	**L30H**	**L40E**
Single	√	√	
Multiple unseen	√		√

the step limit. We measure the difficulty of a game by its *level* (the quest length, i.e. the number of steps required by an optimal agent to solve the game) and *difficulty mode* (the existence of distracting rooms leading to the dead end). A game is considered as easy if there's no distracting room (e.g., "L30E"), otherwise hard (e.g., "L30H"). We consider two different experiment settings for the synthetic games: the single game setting and the multiple unseen games setting. Similar to the RL settings in those Atari games [27], we train and evaluate the agent on a same game in the single game setting. In the setting of multiple unseen games, we aim to evaluate the agent's generalizability towards unseen games, that we build two non-overlapping training and evaluating game sets. Table 4.1 shows the details of the synthetic games and their settings. For the man-made games, we select a set of 15 games from the Jericho platform [85], and consider the single game setting.

4.4.2 Baselines

For the synthetic games, we build our agent based on LSTM-DRQN [226], and consider following baselines:

- LSTM-DRQN [226], which is our backbone model. The representation generator is built with LSTM.
- CNN-DRQN, which is modified from CNN-DQN [220]. The representation generator is built with CNNs.
- Trans-DRQN, which modifies LSTM-DRQN by replacing the LSTM with the vanilla transformer encoder.

- Trans-v-DRQN, which is our proposed agent. The representation generator is built with the modified transformer.

For the man-made games, we build our agent based on DRRN [88], and consider following baselines:

- DRRN [88], which is our backbone model. It's a choice-based agent that the set of admissible actions is assumed to be available at each time step.
- TDQN [85], which extends LSTM-DQN with template-based action space.
- KG-A2C [7], which takes advantage of the knowledge graph-based observation to address the partial observability.
- Trans-v-DRRN, which is our proposed model with the modified transformer as the state representation generator.

4.4.3 Implementation Details

For the synthetic games, we follow similar parameter settings of LSTM-DRQN [226] to implement the models. The transformer architecture is modified based on Pytorch's official example, with the encoder being simplified. Our Trans-v-DRQN consists of $L = 4$ transformer blocks. We set the MHA part as single-head, and set FFN part as a linear layer with 100 hidden units. All models adopt DRQN [87] as the action scorer. Besides, we use reward shaping to encourage exploration - the agent will be assigned 1.0 additional reward if it encounters a state for the first time.

For the man-made games, we follow similar parameter settings of DRRN implemented by [85]. We replace the 3 GRU encoders, which are used for processing the description of current location, the player's inventory and the feedback of previous action, with a shared transformer-based encoder with 4 blocks, 4 heads in the MHA part, and 128 hidden units in the FFN part.

4.4.4 Training Details

For the synthetic games, we set one episode as 50 time steps. We run multiple environments in parallel, and denote the model finishing an episode on all the environments as an epoch. For the

single game setting, we generate 10 environments for a same game. For the multiple unseen games setting, we use 200 training games with one environment per game, and use another 20 games for testing. We apply the ϵ-greedy approach with ϵ being annealed from 1 to 0.2 in 1000 epochs (50 epochs) for the single game setting (multiple unseen games setting). We use the prioritized experience replay with buffer size 5000 and updating batch size 32. We update the models using the Adam optimizer, with learning rate being set as 0.001.

For the man-made games, we set one episode as 100 time steps. Similar to the baselines, we conduct parameter tuning on the game "zork1". We generate 8 environments for each single game. We apply the experience replay with buffer size 500,000 and updating batch size 64. The models are trained for 100,000 steps, where the optimizer is same to that of the agents for the synthetic games.

4.4.5 Evaluation Metrics

We measure the performance using the sum of collected rewards in an episode. Besides, for the single game setting in the synthetic games, we also compare the training epochs required to solve the games (i.e., achieving the maximum reward sum 1.0).

4.5 Results and Discussions

4.5.1 Synthetic Games: Single Game Setting

Fig. 4.4 shows models' performance in synthetic games under the single game setting. The LSTM-DRQN baseline learns slowly, while the other models have higher learning speed. Table 4.2 compares the training epochs required to solve the games, that our Trans-v-DRQN model shows high sample efficiency. In hard games where there exist distracting rooms, our modified transformer helps to extract the navigational information (e.g., "the kitchen is connected to the living room") from the state, yielding more obvious advantage.

We then systematically study the contribution of the modifications. Table 4.3 shows the ablation results, where "X" denotes that the modification "X" is removed from our model. The weight sharing modification contributes the most, that removing it leads to most significant performance

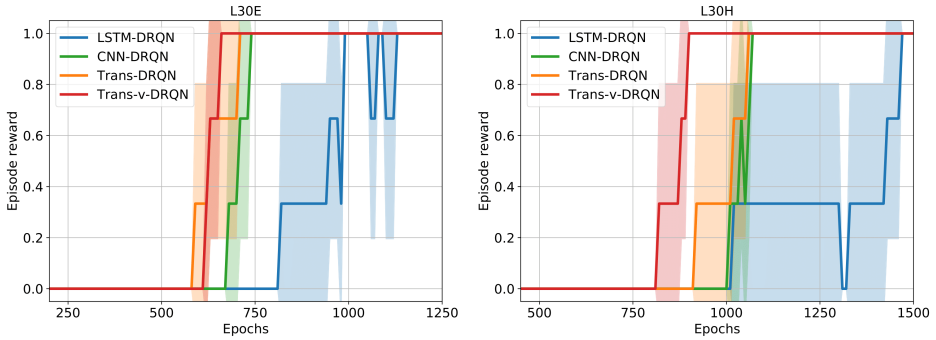


FIGURE 4.4: The models' performance in synthetic games under the single game setting. The shaded area indicates standard deviations over 3 independent runs.

TABLE 4.2: The number of epochs required to solve the single games.

Model	L30E	L30H
LSTM-DRQN	920.00 ± 72.57	1306.67 ± 203.36
CNN-DRQN	710.00 ± 24.49	1040.00 ± 24.49
Trans-DRQN	643.33±49.89	1000.00 ± 58.88
Trans-v-DRQN	**636.67 ± 17.00**	**866.67 ± 33.99**

TABLE 4.3: The performance of models with different modifications.

Model	L30H
Trans-DRQN	1000.00 ± 58.88
Trans-v-DRQN (full)	**866.67 ± 33.99**
-gate	920.00 ± 29.44
-gate -reorderLN	940.00 ± 58.88
-gate -shareW	1050.00 ± 92.74
-gate -reorderLN -shareW	1016.67 ± 37.71

decrease (by comparing "-gate" with "-gate -shareW"). The modification of gating operation also contributes to the performance improvement (by comparing "Trans-v" with "-gate"). The modification of layer normalization reordering contributes the least, that it only slightly improves the performance when being used together with the modification of weight sharing (by comparing "-gate -reorderLN" with "-gate"). Furthermore, the performance even decreases when applying this modification alone (by comparing "-gate -shareW" with "-gate -reorderLN -shareW").

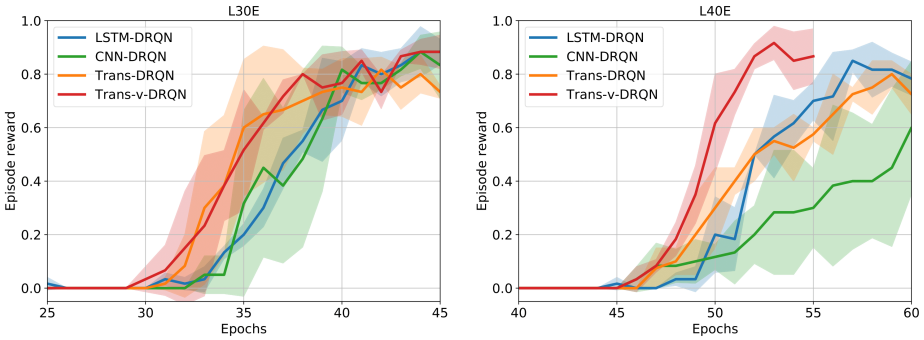


FIGURE 4.5: The models' performance in synthetic games under the multiple unseen games setting.

TABLE 4.4: The maximum rewards obtained in synthetic games under the multiple unseen games setting.

Model	L30E	L40E
LSTM-DRQN	0.88 ± 0.09	0.85 ± 0.04
CNN-DRQN	0.88 ± 0.06	0.60 ± 0.25
Trans-DRQN	0.82 ± 0.05	0.80 ± 0.05
Trans-v-DRQN	$\mathbf{0.88 \pm 0.06}$	$\mathbf{0.92 \pm 0.06}$

4.5.2 Synthetic Games: Multiple Unseen Games Setting

Fig. 4.5 shows models' performance in synthetic games under the multiple unseen games setting. In "L30E", the transformer-based models (Trans-DRQN, Trans-v-DRQN) show higher sample efficiency, that they make progress faster than the other two baselines. However, as the training goes on, the Trans-DRQN baseline starts to degrade and ends up with worst performance, which is possibly due to overfitting. Instead, our Trans-v-DRQN, which is with the modified transformer encoder, is more robust in solving the unseen games. In "L40E", which requires more steps to solve, its advantage is more obvious: while its performance exceeds 0.8 within 55 epochs, the other models require more training epochs and end up with worse result. We also compare the the maximum rewards that can be obtained by the models. Table 4.4 shows the result, that Trans-v-DRQN outperforms the other baselines.

TABLE 4.5: The performance of models on man-made games.

Game	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{V}\rvert$	TDQN	KG-A2C	DRRN	Trans-v-DRRN	MaxReward
acorncourt	151	343	1.6	0.3	10	**10**	30
adventureland	156	398	0	0	20.6	**25.6**	100
affilicted	146	762	1.3	-	2.6	2.0	75
detective	197	344	169	207.9	197.8	**288.8**	360
enchanter	290	722	8.6	12.1	20.0	**20.0**	400
library	173	510	6.3	14.3	17	**17**	30
ludicorp	187	503	6	**17.8**	13.8	16	150
pentari	155	472	17.4	**50.7**	27.2	34.5	70
reverb	183	526	0.3	-	8.2	**10.7**	50
spellbrkr	333	844	18.7	21.3	37.8	**40**	600
temple	175	622	7.9	7.6	7.4	**7.9**	35
tryst205	197	871	0	-	9.6	**9.6**	350
zork1	237	697	9.9	34	32.6	**36.4**	350
zork3	214	564	0	0.1	**0.5**	0.19	7
ztuu	186	607	4.9	9.2	**21.6**	4.8	100

4.5.3 Jericho-supported Games

Table 4.5 compares the performance of models on man-made games. In 10 out of all 15 games, our Trans-v-DRRN surpasses all the other models. In comparison to the building backbone DRRN, Trans-v-DRRN brings further improvements in eight games, and achieves comparable performance in another four games. Fig. 4.6 shows the learning curve of the game "zork1". At the early stage of training, Trans-v-DRRN learns a little slower than the backbone model. However, it starts to learn the solving strategy at about 20,000 updating steps, and eventually outperforms all other baselines. In addition, it can be observed that the transformer architecture helps to overcome the fluctuation problem encountered by the backbone model, leading to better performance at the end of learning.

4.6 Conclusion

In this chapter, we exploited the potentiality of the transformer architecture for building state representations in solving the language-conditional reinforcement learning tasks. We designed an adaptable transformer-based representation generator equipped with three modifications. We first conducted layer normalization reordering within the blocks. Then we shared weights among the blocks. Finally, we applied the gate aggregation between the blocks. We empirically validated our

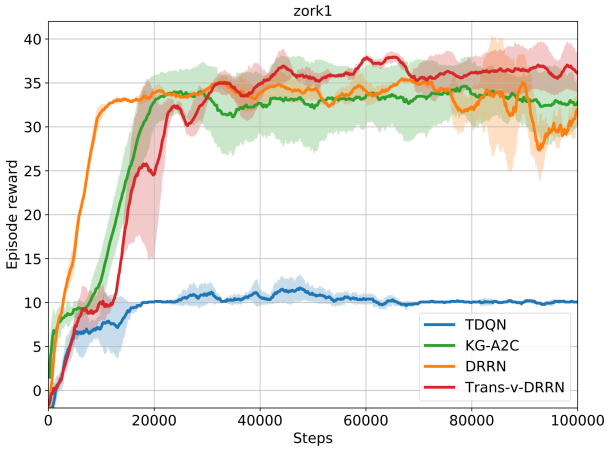


FIGURE 4.6: The models' learning curves on the game "zork1".

method on both synthetic and man-made text-based games with different settings. The proposed method showed higher sample efficiency in solving single synthetic games, better generalizability in solving unseen synthetic games, and better performance in solving complex man-made games.

With respect to the limitations & future directions, currently we only investigate generalizability on simple CoinCollector games, and we would like to extend this study towards more complex games. While current model is trained from scratch using largely simplified transformer architecture, in the future we would like to integrate our model with pre-trained language models [59, 122] to obtain better representation learning ability, thus further exploiting its potential.

Chapter 5

Stacked Hierarchical Attention Mechanism

In this chapter, we study the reasoning process in the language-conditional reinforcement learning. The reasoning ability enables the agent to generate the actions with the support of an explainable inference procedure. To achieve this ability, we propose an agent featured with the stacked hierarchical attention mechanism. Through exploiting the structure of the knowledge graph, this agent is able to explicitly model the reasoning process. Our agent demonstrates effectiveness on a range of man-made text-based games.

5.1 Introduction

While existing studies have been conducted with the aim of building representations from text observations [7, 8, 152] and reducing the combinatorial action space [85, 227], few of them addresses the reasoning process, which we believe that the language-conditional RL agent could benefit from. The human beings are inherently armed with the reasoning ability, that they are able to interpret the decision making process through composing the supporting facts from the context and the knowledge base [32, 114], and reuse the learnt knowledge in the future [209]. For the RL agent, such reasoning ability could be acquired through exploiting the knowledge graphs. Some RL agents have utilized the KG-based observation in the partially observable scenarios, such as the

text-based games [2, 7, 8]. However, the knowledge graph's potential for reasoning is overlooked, and narrowed by two issues [50, 104]. Firstly, most agents consider only one graph, which is hard to maintain the fine-grained information, such as the historical information and different types of object relationship. Secondly, existing agents simply concatenate the inputs (e.g., the KG-based observations and the textual observations) to build the state representation, ignoring the benefits of exploiting the multimodality.

In this work, our aim is to enhance the agent with the reasoning ability in solving language-conditional RL tasks. We design an agent, **S**tacked **H**ierarchical **A**ttention with **K**nowledge **G**raphs (SHA-KG)[1], which is able to conduct multi-step and multl-level reasoning process. We first make the agent aware of the relation and temporal information through dividing the KG as sub-graphs by their semantic meanings. Then, we design a stacked hierarchical attention module to encode the multi-modal observations as the state representation.

Our contributions are summarized as following aspects:

1. Our research is a first step to explore the reasoning ability in language-conditional reinforcement learning.
2. We introduce the reasoning process in decision making through considering a knowledge graph as multiple sub-graphs.
3. We design an RL agent, which is equipped with the stacked hierarchical attention mechanism for reasoning.
4. We empirically validate the effectiveness of our agent in a set of man-made games. Our proposed agent achieves favorable performance in compason with the state-of-the-art baselines.

5.2 Related Work

5.2.1 Reinforcement Learning Agents for Solving Text-based Games

As demonstrated in Chapter 2, existing works have augmented the RL agent with the knowledge graphs, which could be built and updated based on the textual observations. The KGs have been

[1]We release the code at `https://github.com/YunqiuXu/SHA-KG`.

leveraged to reduce action space [7, 8], handle partial observability [7, 8, 230], and improve generalizability [2, 9]. However, the KG's potential for reasoning is seldom addressed. While some works [148] studied commonsense reasoning through incorporating large-scale knowledge base [185], we reduce the usage of such external knowledge by constructing the graph upon the domain information. Furthermore, going beyond the simple synthetic games, we study the reasoning ability in more complex scenarios.

5.2.2 Attention Mechanism

The studies on the attention mechanism have been conducted in areas such as neuroscience, artificial intelligence, and cognitive science [129]. For RL tasks with pixel-based observations, some works have been utilized the attention mechanism for interpreting the action selection [77, 146]. For language-conditional RL, some KG-based agents have adopted the attention mechanism in building state representations. For example, the KG-A2C [7] encoded the graph-based observation via the Graph Attention Networks (GATs) [198]. Some supervised learning works have also employed the attention mechanism to aggregate multi-modal inputs [98, 99, 113, 114, 132, 137]. In our work, we design a new attention mechanism for the purpose of constructing the state representation from the multi-modal observations.

5.2.3 Reasoning upon the KGs

The knowledge graph has been widely applied to provide structural information and the commonsense reasoning ability. While a lot of works have been proposed in domains such as the recommendation systems [201, 203, 213, 232] and question answering tasks [25, 60, 127, 235], we are the first to exploit the KGs to equip the agents with the reasoning ability in solving the language-conditional RL tasks.

5.3 Problem Statement

Compared with the synthetic games [53], the man-made text-based games [85] are much more complex, making them even challenging for human players. In order to solve these games, an agent

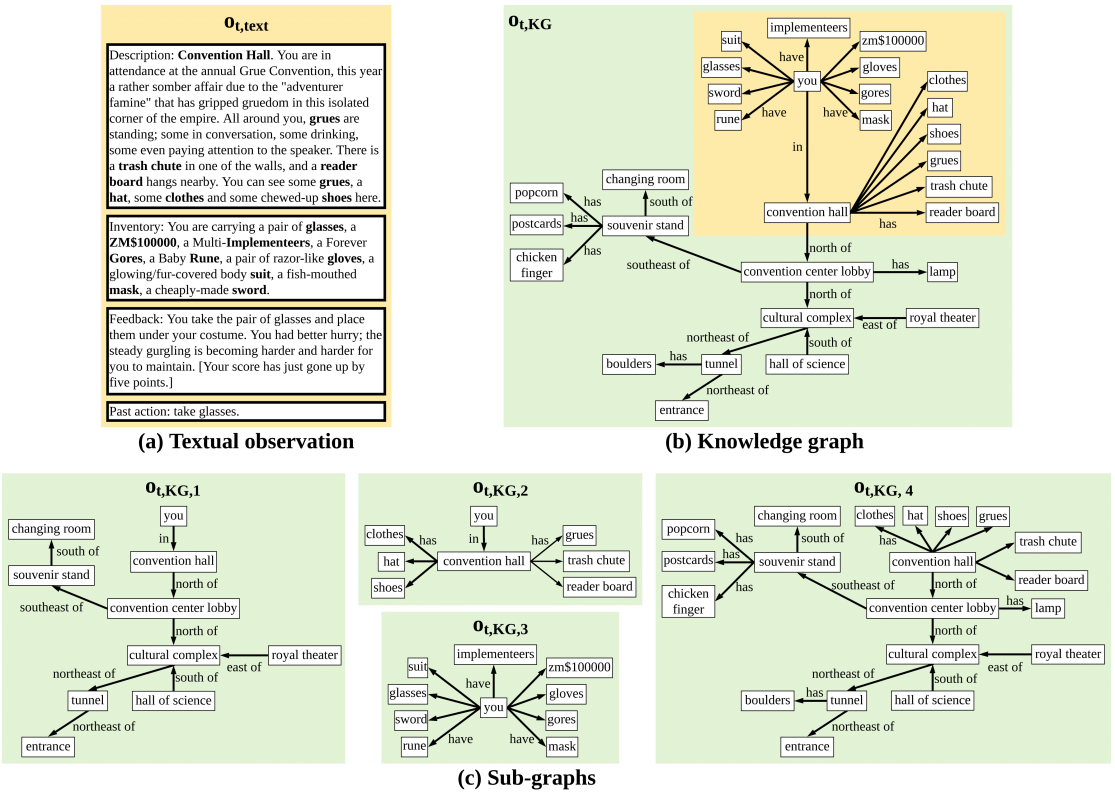


FIGURE 5.1: (a) $o_{t,\text{text}}$ in our work, which consists of four parts. (b) The KG-based observation $o_{t,\text{KG}}$, where those derived from current $o_{t,\text{text}}$ are in yellow. (c) The sub-graphs.

is required to make automatic responses based on some textual information to achieve specific goals, such as finding the treasure and escaping from the dungeon. The input s_t[2] consists of three components: a textual observation $o_{t,\text{text}}$, a knowledge graph-based observation $o_{t,\text{KG}}$, and a collected raw score $o_{t,\text{score}}$. As shown in Fig. 5.1 (a), the $o_{t,\text{text}}$ can be further divided as four parts: the description of the current environment status $o_{t,\text{desc}}$, the objects collected within the inventory $o_{t,\text{inv}}$, the action chosen in the past time step a_{t-1}, and the feedback $o_{t,\text{feed}}$ after executing a_{t-1}. $o_{t,\text{text}}$ and $o_{t,\text{score}}$ represent the current information. $o_{t,\text{KG}}$ helps to address the partial observability through recording the historical knowledge. As shown in 5.1 (b), the knowledge graph $o_{t,\text{KG}}$ is updated using the triples extracted from $o_{t,\text{text}}$:

$$o_{t,\text{KG}} = \text{GraphUpdate}(o_{t-1,\text{KG}}, o_{t,\text{text}}) \tag{5.1}$$

We detail the process about the knowledge graph construction in Sec. 5.5.2.

[2]Note that we should not regard s_t as the environment state, which is not provided to the agent.

5.4 Methodology

5.4.1 Sub-graph Division

Different from most existing works [2, 7, 8, 230], which consider only one KG, we divide the knowledge graph as a set of sub-graphs for the purpose of introducing the relational-awareness and temporal-awareness. We get inspiration from the heterogeneous graph [204, 231], where the nodes / edges within a graph may have different types. We first classify the edge types. For example, we can treat "$Have$" and "$West\ of$" as different categories. Then, based on the edge types, multiple sub-graphs will be constructed with the relational awareness. We consider the sub-graph division in the level of graph semantics. That is, each sub-graph may contain multiple types of edges, and edges with the same type may appear in different sub-graphs. Since the single KG can not distinguish the current information from the global information, we further construct sub-graphs with / without the historical information, thus introducing the temporal awareness to the KG. For example, we can build a sub-graph from $o_{t,\text{text}}$ only, and build another sub-graph from $o_{t,\text{text}}$ and $o_{t-1,\text{KG}}$. Fig. 5.1 (c) displays some sub-graphs derived from $o_{t,\text{KG}}$. For more details of the sub-graph division, please refer to Section 5.5.2.

The full KG can be regarded as the union of all m sub-graphs:

$$o_{t,\text{KG}} = o_{t,\text{KG,1}} \cup o_{t,\text{KG,2}} \ ... \ \cup o_{t,\text{KG,m-1}} \cup o_{t,\text{KG,m}} \tag{5.2}$$

Through dividing the KG as sub-graphs, a two-level hierarchy is established upon the observations [218, 236]. In the high level, the global information is captured by the full knowledge graph. In the low level, the relational as well as the temporal information is modeled by different sub-graphs.

5.4.2 Stacked Hierarchical Attention

Motivated by the Visual Question Answering (VQA) tasks [113, 132], we design a stacked hierarchical attention mechanism for effective aggregation among the textual observation, score and KG-based observation. Considering a query and the context, the query representation will first be constructed from one modal (or two), then be used for obtaining the attention values across another modal (the context). Fig. 5.2 provides an overview of our proposed two-level encoder, where the

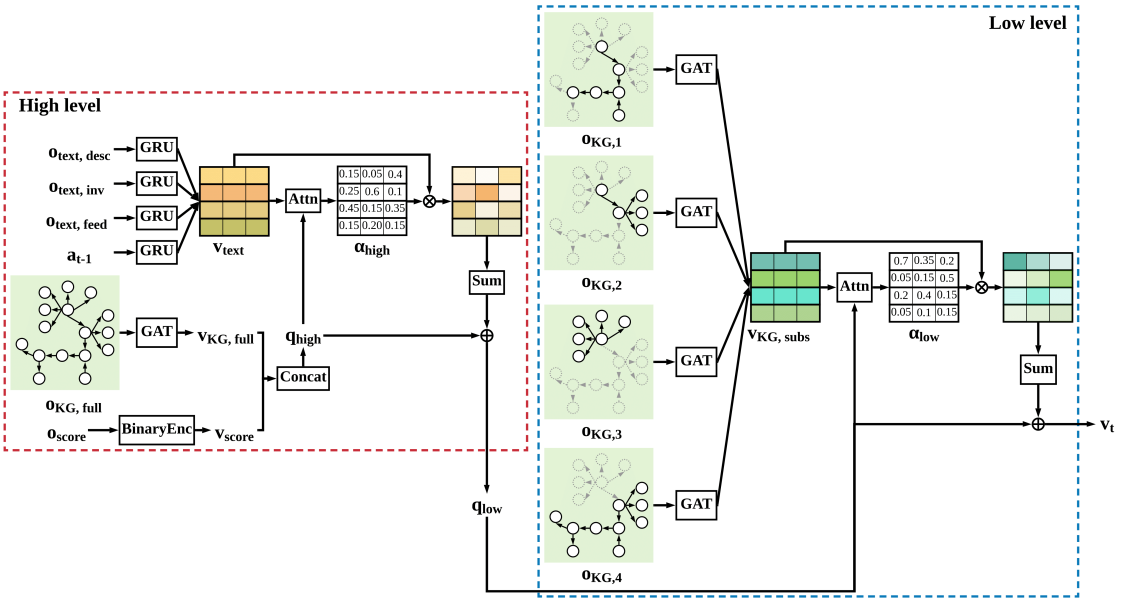


FIGURE 5.2: Overview of the SHA-KG's encoder.

subscript "t" is omitted during encoding, and $o_{\text{KG,full}}$ is the full knowledge graph before sub-graph division. In the high level, the query representation is built upon the knowledge graph and the score, then multiple groups of attention values will be obtained across the modal of textual observations. The output of the high level is used to build the query representation in the low level. The low level attention values will then be obtained through querying across the modal of sub-graphs.

High-level encoding We use a similar way of KG-A2C [7] to obtain the graph representation $\boldsymbol{v}_{\text{KG, full}} \in \mathbb{R}^{d_{\text{KG}}}$ - $o_{\text{KG,full}}$ will be first processed via GATs [198], then aggregated using fully-connected layers. For the score, we obtain the representation $\boldsymbol{v}_{\text{score}} \in \mathbb{R}^{d_{\text{score}}}$ through binary encoding. While previous works [7, 85] obtain the state representation via concatenating all observational vectors, we build the query vector $\boldsymbol{q}_{\text{high}} \in \mathbb{R}^{d_{\text{high}}}$ by concatenating $\boldsymbol{v}_{\text{KG, full}}$ and $\boldsymbol{v}_{\text{score}}$ followed by a linear layer:

$$\boldsymbol{q}_{\text{high}} = W_{\text{Init}}\text{concat}(\boldsymbol{v}_{\text{KG, full}}, \boldsymbol{v}_{\text{score}}) + b_{\text{Init}} \tag{5.3}$$

where $W_{\text{Init}} \in \mathbb{R}^{d_{\text{high}} \times (d_{\text{KG}} + d_{\text{score}})}$ denotes the weight, and $b_{\text{Init}} \in \mathbb{R}^{d_{\text{high}}}$ denotes the bias. The different parts of the textual observation will be encoded separately using c different GRUs, where c denotes the number of parts [3]. Then the c encoded parts will be stacked as the representation vector

[3] As discussed in Section 5.3, c is 4 in this work.

$\boldsymbol{v}_{\text{text}} \in \mathbb{R}^{d_{\text{high}} \times c}$, which can be treated as either the different regions of an image, or a multi-channel image representation. The attention value is computed in channel-wise. Similar operations can be observed in SCA-CNN [48], where one single attention value is assigned for each channel. However, in order to extract more fine-grained knowledge, we make a difference by assigning each channel with multiple groups of attention. For each position along the channel, one group of attention values is computed as:

$$\boldsymbol{\alpha}_{\text{high}} = \text{softmax}(W_{\text{A,high}} \boldsymbol{h}_{\text{high}} + b_{\text{A,high}}) \tag{5.4}$$

where the intermediate representation $\boldsymbol{h}_{\text{high}}$ is computed as:

$$\boldsymbol{h}_{\text{high}} = \tanh(W_{\text{I,high}} \boldsymbol{v}_{\text{text}} \oplus (W_{\text{Q,high}} \boldsymbol{q}_{\text{high}} + b_{\text{Q,high}})) \tag{5.5}$$

$W_{\text{I,high}} \in \mathbb{R}^{d_{\text{high}} \times d_{\text{high}}}$, $W_{\text{Q,high}} \in \mathbb{R}^{d_{\text{high}} \times d_{\text{high}}}$ and $W_{\text{A,high}} \in \mathbb{R}^{d_{\text{high}} \times d_{\text{high}}}$ are learnable weights. $b_{\text{Q,high}} \in \mathbb{R}^{d_{\text{high}}}$ and $b_{\text{A,high}} \in \mathbb{R}^{d_{\text{high}}}$ denote biases. The operation of computing multiple groups of attention is similar to that we first divide $\boldsymbol{v}_{\text{text}}$ as d_{high} vectors $\boldsymbol{v}_{\text{text,sub}} \in \mathbb{R}^{1 \times c}$, then assign single attention in channel-wise for each of these vectors. The resulted attention vector $\boldsymbol{\alpha}_{\text{high}} \in \mathbb{R}^{d_{\text{high}} \times c}$, which indicates the attentive focus for each part of the textual observation, will be finally aggregated with the query vector. Motivated by the recent progress in the attention mechanism [74, 217, 225], we propose to update the query vector iteratively, thus enabling multi-step reasoning. $\boldsymbol{v}_{\text{text}}$ is first multiplied with $\boldsymbol{\alpha}_{\text{high}}$ using dot-product, followed by the channel-wise summing operation. The resulted vector will then be aggregated with $\boldsymbol{q}_{\text{high}}$ for obtaining the query vector $\boldsymbol{q}_{\text{low}} \in \mathbb{R}^{d_{\text{high}}}$:

$$\boldsymbol{q}_{\text{low}} = \boldsymbol{q}_{\text{high}} + \sum_{i}^{c} \boldsymbol{\alpha}_{\text{high},i} \odot \boldsymbol{v}_{\text{text},i} \tag{5.6}$$

Low-level encoding We use a similar way of the high level encoding to compute the attention. Firstly, we build the query vector using the output of the high-level encoding, where the linear mapping operation is performed to make sure $\boldsymbol{q}_{\text{low}} \in \mathbb{R}^{d_{\text{low}}}$. Then sub-graphs are encoded by different graph encoders, and stacked to form the representation vector $\boldsymbol{v}_{\text{KG}} \in \mathbb{R}^{d_{\text{low}} \times m}$. Then we compute the multiple groups of attention $\boldsymbol{\alpha}_{\text{low}} \in \mathbb{R}^{d_{\text{low}} \times m}$ as:

$$\boldsymbol{\alpha}_{\text{low}} = \text{softmax}(W_{\text{A,low}} \boldsymbol{h}_{\text{low}} + b_{\text{A,low}}) \tag{5.7}$$

where the intermediate vector $\boldsymbol{h}_{\text{low}}$ is computed as:

$$\boldsymbol{h}_{\text{low}} = \tanh(W_{\text{I,low}}\boldsymbol{v}_{\text{KG}} \oplus (W_{\text{Q,low}}\boldsymbol{q}_{\text{low}} + b_{\text{Q,low}})) \tag{5.8}$$

$W_{\text{I,low}} \in \mathbb{R}^{d_{\text{low}} \times d_{\text{low}}}$, $W_{\text{Q,low}} \in \mathbb{R}^{d_{\text{low}} \times d_{\text{low}}}$ and $W_{\text{A,low}} \in \mathbb{R}^{d_{\text{low}} \times d_{\text{low}}}$ are learnable weights. $b_{\text{Q,low}} \in \mathbb{R}^{d_{\text{low}}}$ and $b_{\text{A,low}} \in \mathbb{R}^{d_{\text{low}}}$ denote biases. Finally, the state representation vector $\boldsymbol{v}_t \in \mathbb{R}^{d_{\text{low}}}$ is obtained by attentively aggregating $\boldsymbol{q}_{\text{low}}$ and $\boldsymbol{v}_{\text{KG}}$:

$$\boldsymbol{v}_t = \boldsymbol{q}_{\text{low}} + \sum_{i}^{m} \boldsymbol{\alpha}_{\text{low},i} \odot \boldsymbol{v}_{\text{KG},i} \tag{5.9}$$

5.4.3 Action Selection and Model Optimization

Arbitrary approaches could be used for selecting the action, such as recurrent decoding [7] and template-based scoring [85]. In our work, the action is obtained through a two-GRU decoding process. Firstly, a GRU is used to select a template u conditioned on $\boldsymbol{v}_t$ from a set of template candidates $\mathcal{T}$. Then, another GRU is executed for k times to select objects $\{p_i \in \mathcal{P}, i \in [1, ..., k]\}$ recurrently, where the selecting probability p_i is conditioned on both the state representation $\boldsymbol{v}_t$ and the previous prediction (u or p_{t-1}). The set of object candidates $\mathcal{P}$ is defined as the intersection of two sets: the set of interactive objects extracted from $o_{\text{KG,full}}$, and the set of vocabulary $\mathcal{V}$. Finally, we combine the predicted objects and the template as an action a_t. Regarding the model optimization, we follow [7] to train the agent using the Advantage Actor Critic (A2C) method [143].

5.5 Experiments

We conduct experiments on 20 man-made text-based games supported by the Jericho platform [85]. We consider following three research questions:

1. Whether the sub-graph division as well as the stacked hierarchical attention helps to introduce the reasoning process?

2. Whether equipping the agent with the reasoning ability helps it to solve the complex text-based games?

3. How to interpret the reasoning process in decision making?

5.5.1 Baselines

Our model and the baselines are shown as follows:

- NAIL [86], which is a non-RL agent with pre-trained language models and hand-crafted rules.
- DRRN [85, 88], which is a choice-based agent, where a set of valid actions will be provided per time step.
- TDQN [85], which is a template-based agent.
- KG-A2C [7], which extends TDQN with the knowledge graph-based observation.
- SHA-KG, which is our proposed model featured with the stacked hierarchical attention.

5.5.2 Experimental Setup

Graph construction We use Stanford Open Information Extraction (OpenIE) [16] to extract the triples from the textual observations. In addition, we adopt two simple rules from [7]: 1) In the current observation, we link the interactive objects within the inventory to the node "you", and link other interactive objects to current location. 2) We infer the location connectivity from the navigational actions (e.g., "go east"). There are 11 edge types: "north", "south", "east", "west", "northeast", "northwest", "southeast", "southwest", "in" (indicating the player's location), "have" (indicating an object's location, including the player's inventory), and "is" (indicating an object's status). We define four graph types for sub-graph division. $o_{\text{KG},1}$ denotes the connectivity of visited locations, $o_{\text{KG},2}$ denotes the interactive objects at current location, $o_{\text{KG},3}$ denotes the collected items, and $o_{\text{KG},4}$ contains those not connected to the node "you". $o_{\text{KG},1}$ and $o_{\text{KG},4}$ represent both current and historical knowledge, while $o_{\text{KG},2}$ and $o_{\text{KG},3}$ represent the current knowledge only. We leave the automatic graph partitioning methods as future work.

Training details We implement SHA-KG upon the KG-A2C with similar hyper-parameter settings [7]. We set the node dimensionality in GATs as 25, the query dimensionality in the high level

TABLE 5.1: The main result in 20 games.

Game	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{V}\rvert$	NAIL	DRRN	TDQN	KG-A2C	SHA-KG (Ours)	MaxR
acorncourt	151	343	0	**10**	1.6	0.3	1.6	30
balances	156	452	**10**	**10**	4.8	**10.0**	**10.0**	51
detective	197	344	136.9	197.8	169	207.9	**308.0**	360
dragon	177	1049	**0.6**	-3.5	-5.3	0	0.2	25
enchanter	290	722	0	**20.0**	8.6	12.1	**20.0**	400
inhumane	141	409	0.6	0	0.7	3	**5.4**	300
jewel	161	657	1.6	1.6	0	**1.8**	**1.8**	90
library	173	510	0.9	**17**	6.3	14.3	15.8	30
ludicorp	187	503	8.4	13.8	6	**17.8**	**17.8**	150
pentari	155	472	0	27.2	17.4	50.7	**51.3**	70
reverb	183	526	0	8.2	0.3	7.4	**10.6**	50
sorcerer	288	1013	5	20.8	5	5.8	**29.4**	400
spellbrkr	333	844	**40**	37.8	18.7	21.3	**40.0**	600
spirit	169	1112	1	0.8	0.6	1.3	**3.8**	250
temple	175	622	7.3	7.4	**7.9**	7.6	**7.9**	35
tryst205	197	871	2	**9.6**	0	6.7	6.9	350
zenon	149	401	0	0	0	**3.9**	**3.9**	350
zork1	237	697	10.3	32.6	9.9	34	**34.5**	350
zork3	214	564	**1.8**	0.5	0	0.1	0.7	7
ztuu	186	607	0	21.6	4.9	9.2	**25.2**	100

encoding d_{high} as 50, and the query dimensionality in the low level encoding d_{low} as 50. We define an episode as 100 interaction steps. For each game, we train a single agent for for 10^6 interaction steps. For every 8 interaction steps, we update the model using Adam optimizer with the learning rate being set as 0.003. We implement all baseline models based on their original paper [7, 85, 86]. We measure the performance using the mean score of the last 100 training episodes.

5.6 Results and Discussions

5.6.1 Main Results

Table 5.1 displays the main result in 20 games. Different games have different sizes of template set $|\mathcal{T}|$ and vocabulary set $|\mathcal{V}|$. **MaxR** is the highest result that could be achieved by the human expert player, which could be treated as the upper bound for this benchmark. Although according to **MaxR**, there is still a long way to go for the game agents to achieve the human-level performance, our SHA-KG achieve promising performance among the baselines. It surpasses the current best

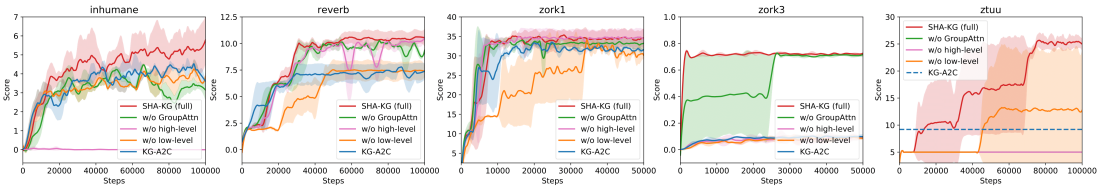


FIGURE 5.3: The result with respect to the update steps for SHA-KG variants with different encoding methods.

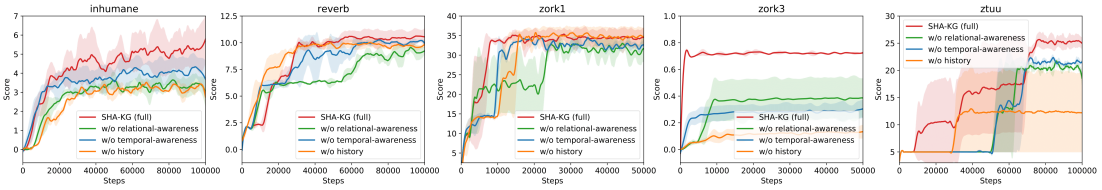


FIGURE 5.4: The result with respect to the update steps for SHA-KG variants with different sub-graphs.

result in 8 games, and obtains equivalent performance of the best-performed agents in another 7 games. In particular, the strength of incorporating the reasoning ability could be validated by comparing SHA-KG with other baselines with the template-based action space, such as TDQN and KG-A2C, that SHA-KG achieves comparable or better result in all of the 20 games. It can also be observed that SHA-KG still can not beat another two baselines, NAIL and DRRN, in 5 games. As a non-RL agent, NAIL is equipped with engineering tricks and pre-defined solving procedures. For example, when entering a new location, it will try to interact with all of the observed objects until reaching the step limit. Although such external knowledge might be helpful in solving some specific games, it is less flexible, leading to worse performance than the RL baselines in most of the games. Another baseline, DRRN, is with a strong assumption that the set of admissible actions are provided per time step. This assumption prevents the agent from wasting time in choosing meaningless actions (e.g., selecting "take knife" when the object "knife" does not exist, or selecting a wrong word combination "eat knife"). Although this assumption is relaxed by using the templates, the challenge of large action space still exists ($\mathcal{O}(\mathcal{TP}^2)$), making it difficult to learn the solving strategy quickly. However, compared with our backbone model KG-A2C, the reasoning ability still brings significant improvement. SHA-KG exceeds DRRN in 6 out of 9 games where KG-A2C is defeated by DRRN. In another 3 games, the performance of SHA-KG is close to that of DRRN.

5.6.2 Ablation Studies

With the aim of studying the contributions of the components, we carry out two branches of ablation studies. We first consider SHA-KG variants with different encoding methods:

- "w/o GroupAttn", where each channel is assigned with a single attention value, i.e., $\boldsymbol{\alpha}_{\text{high}} \in \mathbb{R}^4$, $\boldsymbol{\alpha}_{\text{low}} \in \mathbb{R}^4$.
- "w/o high-level", which does not use the full KG - the query representation is constructed upon $\boldsymbol{v}_{\text{text}}$ and $\boldsymbol{v}_{\text{score}}$, then the attention values for different sub-graphs are computed.
- "w/o low-level", which does not use the sub-graph division - the query representation is constructed upon $\boldsymbol{v}_{\text{KG, full}}$ and $\boldsymbol{v}_{\text{score}}$, then the attention values for different textual observation parts are computed.

Fig. 5.3 compares the result with respect to the update steps, where the result from [7] is represented as the dashed line. In all games, our SHA-KG outperforms, or at least has similar performance to the model variants. This can be explained from the perspective of our two-level attention mechanism, that the high level attention guides the agent to target at important parts of the textual observation, and the low level attention helps it to focus on important sub-graphs. We observe that considering the sub-graphs helps to improve the data efficiency for the variant "w/o high-level" in some games such as "zork1". However, without the full KG this variant is with bad performance in the other games such as "inhumane" and "zork3". In contrast, if we consider the full knowledge graph only, the agent learns slowly (e.g., "w/o low-level" in game "zork1"). Similar to the full model, the variant "w/o GroupAttn" also considers the full KG as well as the sub-graphs. However, through capturing the textual / graph information in a more fine-grained level, the full SHA-KG outperforms "w/o GroupAttn" by a large margin, indicating the effectiveness of computing multiple groups of attention.

We then compare SHA-KG variants with different kinds of sub-graphs:

- "w/o relational-awareness", where $o_{\text{KG,2}}$ (objects within current location) is combined with $o_{\text{KG,3}}$ (collected objects).

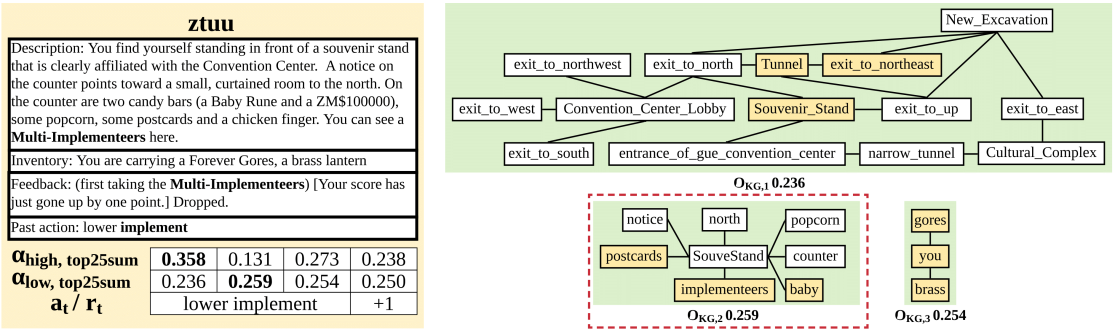


FIGURE 5.5: An example of the reasoning process for game "ztuu".

- "w/o temporal-awareness", where $o_{\text{KG},4}$ is combined with $o_{\text{KG},2}$ and $o_{\text{KG},3}$, respectively. In other words, this variant can not distinguish the historical information from the present information.
- "w/o history", where the historical information is removed from the sub-graphs.

Fig. 5.4 compares the result with respect to the update steps. We observe that in different games, these sub-graph variants have different effects on the agent. For example, in two games "zork1" and "zork3", the variants "w/o relational-awareness" and "w/o temporal-awareness" demonstrate different behaviors. While it is hard to conclude which sub-graph division method has the most significant contribution, we suggest collectively considering different sub-graphs, which leads to the promising performance of the full model.

5.6.3 Interpretability

Besides the quantitative results, we use the attentive focus to interpret the reasoning and decision making process. Although controversial opinions have been raised regarding whether the word-level attention can be used to explain the predictions [101, 210], the proposed attention mechanism is operated in a way closer to the region/channel-level attention, whose effectiveness in explanation has been validated in the RL domains with pixel-based inputs [77, 146]. Since the agent computes multiple groups of attention values (e.g., d_{high} attention values are assigned for each part of the textual observation), we conduct aggregation to facilitate the interpretation. For each "channel" (a part of the textual observation, or a sub-graph), a single attention value is first obtained by summing the top 25 highest group values. Then, the softmax operation is conducted across the channels for

normalization. As displayed in Fig. 5.5 (left), $\boldsymbol{\alpha}_{\text{high,top25sum}}$ corresponds to the textual parts in the high level ($o_{\text{text,desc}}$, $o_{\text{text,inv}}$,$o_{\text{text,feed}}$ and $a_{\text{t-1}}$), and $\boldsymbol{\alpha}_{\text{low,top25sum}}$ corresponds to the sub-graphs in the low level ($o_{\text{KG, 1}}$, $o_{\text{KG, 2}}$, $o_{\text{KG, 3}}$ and $o_{\text{KG, 3}}$). It can be observed that three textual parts containing the word "implement" attract more attention in the high level (i.e., the description of the current environment status $o_{\text{text, desc}}$ followed by the feedback part $o_{\text{text, feed}}$ and the previous action part $a_{\text{t-1}}$). In terms of $\boldsymbol{\alpha}_{\text{low,top25sum}}$, $o_{\text{KG,2}}$, which records the interactive objects at current location, attracts the most attention. By taking the attention focuses from the both levels into consideration, the agent eventually chooses "lower implement", which leads to a "+1" reward. Fig. 5.5 (right) visualizes three sub-graphs and their graph-level attention, where the agent will pay more attention to $o_{\text{KG,2}}$. We also provide the node-level attention, despite the fact that the GATs in SHA-KG are not for computing the attention values, but serving as the graph encoder. In each sub-graph, those with top three largest node-level attention are highlighted in yellow, helping the agent to further refine the information in selecting the actions. To summary, the two-level stacked hierarchical attention, as well as the node-level attention, aids our SHA-KG agent to derive decisions in an efficient and interpretable way.

5.7 Conclusion

In this chapter, we probed the reasoning process in the language-conditional reinforcement learning. We designed the SHA-KG agent, which is empowered with the reasoning ability over multi-modal inputs through two techniques: the knowledge graph division, and the stacked hierarchical attention mechanism. Besides demonstrating the effectiveness of our agent in solving a range of man-made text-based games, we also interpreted how the reasoning process is conducted by SHA-KG in deriving the actions.

With respect to the limitations & future directions, the KG construction in our work is still task-dependent, and we would like to study a more generalizable construction approach. Besides, we only consider some simple sub-graph division approaches, while more types of KGs, such as sub-graphs with fine-grained temporalities, could be investigated in the future.

Chapter 6

Hierarchical Knowledge Graph Agent

In this chapter, we study the generalization problem in language-conditional RL. We consider the knowledge graph-based observation, and address this challenge by designing a two-level hierarchical RL agent. In the high level, we use a meta-policy for task decomposition and subtask selection. Then, in the low level, we use a sub-policy for subtask-conditioned action selection. In a series of 8 game sets with different generalization types and game difficulty levels, our proposed agent enjoys generalizability and yields favorable performance.

6.1 Introduction

In pursuing the strong AI, it is crucial to make agents generalizable to be adapted to different scenarios. In reinforcement learning, nevertheless, the generalization still remains challenging – the agent tends to overfit the training environment, and fails to generalize to new environments [51]. Recently, the TextWorld [53] platform facilitates studying generalizability in language-conditional RL through generating non-overlapping training and testing game sets with customizable domain gaps (vocabulary sets, themes, layouts, complexities, etc.). Most existing works consider two types of generalization: 1) generalization across different games within a same level [8] (e.g., games have same number of rooms, but are different in room connectivity), and 2) generalization across games from a series of multiple difficulty levels [3] (e.g., by varying the number of rooms). While performing well on some relatively easy games, it's hard for the existing agents to achieve

satisfactory results in solving more complex games [2]. Our aim in this work lies in developing agents with generalizability towards both generalization types. In addition, we propose to extend the generalization across games to more challenging settings, where the testing games come from unseen difficulty levels.

Due to long-term temporal dependencies, it might be difficult to learn to solve a whole task (e.g., solving a game). Furthermore, there exists large domain gap between the games (e.g., two cooking games might have completely different recipes), making it difficult to transfer the learnt solving strategy across them. The problem in solving a whole task could be alleviated by treating the task as a sketch of subtasks, which are easier to complete, since they have shorter term of temporal dependencies [12, 157]. Besides, it would be more feasible to solve an unseen task through recomposing the strategies learnt from solving subtasks. Drawn inspiration from such task sketches, we propose to first decompose the task as subtasks, then make decisions conditioned on a subtask. We leverage the Hierarchical Reinforcement Learning (HRL) framework [191] to eliminate the requirement for hand-crafting the task sketches, and improve generalizability through exploiting the compositional nature of language [105].

We develop an RL agent, **H**ierarchical **K**nowledge **Graph**-based **A**gent (H-KGA)[1], which consists of a meta-policy in the high level, and a sub-policy in the low level. The meta-policy will be used for subtask generation and selection, that it first obtains a set of available subtasks identified by language-based goals, and selects one from them. The sub-policy will then be used to select actions conditioned on the selected subtask. Besides the agent, we propose two techniques, the scheduled task sampling and the level-aware replay buffer, to facilitate training in games from multiple levels. The experiments are conducted on 8 sets of text-based games with different levels, and different generalization types. In comparison to the baselines, the proposed method enjoys improved generalizability, which leads to promising performance.

Our contributions are summarized as following three phases:

1. Our research is a first step to address the generalization problem in language-conditional reinforcement learning from the perspective of hierarchical RL.

[1]We release the code at: `https://github.com/YunqiuXu/H-KGA`

2. Upon the knowledge graph-based observationm, we propose a hierarchical agent, which is featured with adaptive subtask decomposition, subtask selection, and subtask-conditioned action selection.

3. We show that our agent improves generalizability in solving multiple sets of games with various difficulty levels.

6.2 Related Work

Existing works have studied how to construct the knowledge graph from the textual observation [7, 147, 216]. Our work, which focuses on improving the generalizability through exploiting the KG-based observation, complements them.

6.2.1 Generalization in Text-based Games

The man-made text-based games are still challenging for the existing RL agents even under the single game setting [219]. Furthermore, it's hard to determine the domain gap between these games, as they vary largely in their themes, vocabularies and logics [9]. The synthetic games [53, 196], instead, provide a more natural way to study generalization - multiple games can be generated with controllable domain gaps. According to previous works, the games in the training set and the testing set either have the same difficulty [8, 147], or come from a mixture of various difficulties [3, 221], or have both settings [2]. In our work, we also consider the multi-difficulty setting. Besides, our work extends this setting towards unseen difficulties - the agent will be tested on games whose levels are not encountered during training. Additionally, we emphasize on enhancing agent's generalizability for games from more difficult scenarios.

6.2.2 Hierarchical Reinforcement Learning

The hierarchical reinforcement learning [57] has been studied in robotic control tasks [149], video games [120, 181, 199], and dialogue systems [161, 170]. However, to the best of our knowledge, our research is one of the initial steps to consider the HRL framework in solving text-based games

with the KG observations. The idea of characterizing a task by language-based specifications has been investigated in instruction following tasks [21, 71]. Such insight has also been considered by some text-based game agents, for the propose of generating quests [6, 11]. In our research, we use the goal to identify a subtask. However, instead of pre-specifying a goal, or directly generating the goal from the observation, we introduce a hierarchical framework, where the goal set generation and goal selection are disentangled. Another work similar to ours is HIN [105], which also considered a meta-policy for goal selection, and a sub-policy for goal-conditioned action selection. However, there are two major differences between their work and ours. Firstly, we study the scenario with language-based observation and action space, while HIN focuses on visual scenarios. Secondly, while the two policies in HIN are trained separately, and the joint training schema is left as a future direction, our framework enables joint training of the policies. We further compare the joint training with the individual training in Sec. 6.6.

6.3 Problem Statement

Different from Chapter 4 and Chapter 5, where the textual observations are used, in this chapter we consider the KG-based observation o_t^{KG}. Besides, we assume the accessibility of the set of admissible actions $A_t \subseteq \mathcal{A}$. Both o_t^{KG} and A_t are provided by the environment for each time step. Fig. 6.1 shows an example of o_t^{KG}.

We aim at improving agent's generalizability for the language-conditional RL tasks. For simplicity, we only consider games having similar themes, but are different in their layouts as well as difficulty levels, so that there's no need for acquiring external knowledge from the pre-trained language models [37, 59]. Taking the cooking theme [53] as an example, the overall objective is to prepare the meal, which is shared by all games. Accomplishing this objective requires the agent to gather all the ingredients, then correctly prepare each of them. A game's layout consists of the connectivity of rooms, and the preparing steps for each ingredient (roast, fry, dice, etc.). A game's difficulty level depends on the recipe (the number of the ingredients, the preparing steps per ingredient, etc.) and the map's complexity (the number of rooms) - if two games come from different difficulty levels, they will naturally have different layouts. We define the training / validation / testing sets to be with multiple games, where the games within a set may have different layouts and / or different difficulty levels. We study two types of generalization: 1) Generalization across seen difficulty

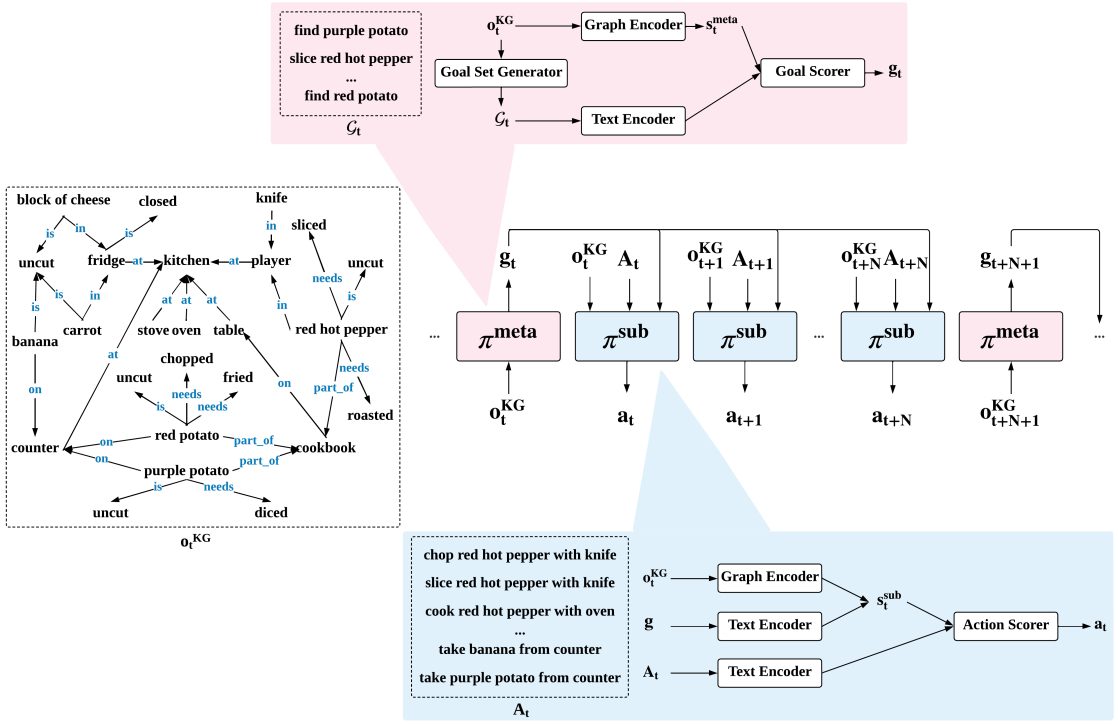


FIGURE 6.1: An overview of the proposed H-KGA, where the high level decision making process is in red (goal set generation and goal selection), and the low level decision making process is in blue (action selection).

levels, where the training games and the testing games belong to a same set of levels, but are different in layouts. 2) Generalization across unseen difficulty levels, where training games and testing games are different in both layouts and levels.

6.4 Methodology

6.4.1 Overview

Fig. 6.1 displays an overview of the proposed H-KGA agent, where the policies are executed in a hierarchy of two levels. In the high level, we consider a meta-policy π^{meta} to get the set of currently available subtasks identified by the goals $\mathcal{G}_t$ from o_t^{KG}, and select a goal $g_t \in \mathcal{G}_t$. Then, based on g and the observation o_t^{KG}, we consider a sub-policy π^{sub} to choose an action $a_t \in A_t$. We omit the subscript "t" for g, since a goal might be picked in the past. Fig. 6.1 shows an example, that once chosen, the same goal g_t will be used by π^{sub} for N steps until being failed or accomplished. The rest of this section is organized as follows: Sec. 6.4.2 demonstrates how to use π^{meta} to get $\mathcal{G}_t$, and

select a goal g_t from $\mathcal{G}_t$. Sec. 6.4.3 illustrates how to use π^{sub} for selecting the action a_t from A_t. And Sec. 6.4.4 illustrates how to train H-KGA under the setting of multi-task learning.

6.4.2 Meta-policy

As illustrated before, decomposing a while task into subtask sketches will be helpful to deal with the problem of long-term temporal dependency, thus reducing the difficulty in solving it [179, 184]. Moreover, if the strategy for solving a subtask can be treated as a skill, through recomposing the learnt skills the agent can also improve its generalizability towards unseen tasks. Therefore, inspired by the HRL framework [120, 191], we consider a hierarchical framework to incorporate such task decomposition. We denote a policy, which is executed in the high level, as meta-policy π^{meta}. This policy will first generate a set of subtasks, and then take one subtask from them. By characterizing a task with its goal, the subtask selection can be transformed into goal selection. Instead of using a state as the goal, we make the goal be in the form of language-based instructions ("dice yellow banana", "roast red apple", etc.), yielding better flexibility and interpretability [15]. Fig. 6.1 (red) shows the overview of π^{meta} , which consists of four parts: the goal set generator, the graph encoder, the text encoder and the goal scorer. We regard the set containing the goals essential for solving a task as $\mathcal{G}$. A goal is regarded as "available" if it does not have any prerequisite goals at the current time step. For example, the goal "cook red potato" is not available in Fig. 6.1, as the agent should accomplish another one "find red potato" first. Regarding the goal set generator, we consider two usages: 1) obtaining the set of goals that are currently available $\mathcal{G}_t \subseteq \mathcal{G}$, and 2) checking if a goal is accomplished. Different approaches can be used for implementing the goal set generator, such as functional programs, human supervisors and pre-trained language models [105]. We consider a non-learning-based goal set generator for obtaining $\mathcal{G}_t$, please refer to Sec 6.5.3 for details.

After obtaining the available goal set $\mathcal{G}_t$, π^{meta} will then be used for goal selection. We first obtain the high level state representation $\boldsymbol{s}_t^{\text{meta}}$ from o_t^{KG} via the graph encoder. In our work, the graph encoder is implemented upon the Relational Graph Convolutional Networks (R-GCNs) [173], that both nodes and edges will be taken into consideration. We then obtain a stacked goal representations from $\mathcal{G}_t$ via the text encoder. Since we use short texts to describe a goal, it's sufficient to implement the text encoder based on a simple single-block transformer [197]. Regarding the goal scorer, we score the goals in a manner similar to DRRN [88]: we first pair the state representation $\boldsymbol{s}_t^{\text{meta}}$ with

the representation of each goal candidate, then process these representation using fully-connected layers. We follow the Q-learning setting to treat the score as Q value, that the candidate with the maximum score will be chosen.

Following the Semi-Markov Decision Process (SMDP) [191], π^{meta} will be re-executed once a goal is accomplished / failed. π^{meta} receives the environmental reward r_t^{env}. For a high level transition, the reward is defined as the sum of environmental rewards:

$$r^{\text{meta}} = \sum_{i=1}^{T} r_{t+i}^{\text{env}} \tag{6.1}$$

where T is the number of time steps spent on accomplishing g_t.

6.4.3 Sub-policy

The sub-policy π^{sub} follows the setting of goal-conditioned RL [107], where a_t is conditioned on both o_t^{KG} and g. As shown in Fig. 6.1, the architecture of π^{sub} (blue) is similar to π^{meta}, except that the state $\boldsymbol{s}_t^{\text{sub}}$ is constructed based on both o_t^{KG} and g. The graph encoder and text encoder π^{sub} can be implemented independently, or re-using those in π^{meta}. As this work does not address the challenges of large action space, we assume the availability of a subset of currently admissible actions $A_t \subseteq \mathcal{A}$, where an action is "admissible" if executing it doesn't result in meaningless feedback such as "I don't understand this command". The action scoring process is conducted in a way similar to the scoring process in π^{meta}, that each action candidate $a_i \in A_t$ is pair with the low level state representation $\boldsymbol{s}_t^{\text{sub}}$, then the action scores will be computed through fully-connected layers.

Depending on whether a goal is accomplished, π^{sub} receives binary reward $r_t^{\text{goal}} \in \{r_{\text{min}}, r_{\text{max}}\}$. We reuse the goal set generator to realize such intrinsic reward function. In Fig. 6.1, for instance, if the agent has the goal "find knife" before observing o_t^{KG}, it will be assigned with the maximum reward $r_t^{\text{goal}} = r_{\text{max}}$, since "find knife" is successfully finished at time t. In the complex scenarios, such goal-conditioned binary reward is still inadequate, even though the knowledge graph can be treated as a "map" to guide the agent. For example, some cooking games may have a large number of rooms, that the agent has to spend time on looking for the ingredients, while the intrinsic reward is assigned sparsely in this process. In order to conduct efficient exploration, thus further

improving the performance of π^{sub}, we enhance sub-policy's reward function with the count-based reward shaping [26]. In particular, we adopt the BeBold method [234]. During the training process, we not only count the state visitation within each episode, but also record the accumulated state visitation throughout the whole training process. We compute the reciprocal of the accumulated state visitation for the states before and after executing the action, then regularize their difference through the episodic state visitation:

$$r_{t+1}^{\text{count}} = \max\left(\frac{1}{\mathrm{N}_{\text{acc}}(o_t^{\text{KG}})} - \frac{1}{\mathrm{N}_{\text{acc}}(o_{t+1}^{\text{KG}})}, 0\right) \cdot \mathbb{I}\{\mathrm{N}_{\text{epi}}(o_{t+1}^{\text{KG}}) = 1\} \tag{6.2}$$

where N_{epi} is the episodic state visitation, and N_{acc} is the accumulated state visitation. The $\mathbb{I}$ function returns 1 if o_{t+1}^{KG} is observed for the first time within current episode, otherwise 0. Finally, we combine r_{t+1}^{goal} with r_{t+1}^{count} as the reward for π^{sub}:

$$r_{t+1}^{\text{sub}} = r_{t+1}^{\text{goal}} + \lambda \cdot r_{t+1}^{\text{count}} \tag{6.3}$$

where the coefficient λ is used to balance these two components.

6.4.4 Training Strategies

We train H-KGA via Double DQN [84] with prioritized experience replay [172]. Algo. 1 demonstrates the training process. The training set $\mathcal{D}_{\text{train}}$ consists of games from $\mathcal{L}$ levels, that a game $x \in \mathcal{D}_{\text{train}}$ is sampled per episode (lines 2-22). The goal g is regarded as terminated and will be re-selected by π^{meta} if the agent 1) completes it successfully, 2) fails to accomplish it, or 3) exceeds the total step limit NUM_STEPS. Supposing that the game x belongs to level $l \in \mathcal{L}$, the transition for the high level decision making will be formulated as $\langle o_t^{\text{KG}}, g, r^{\text{meta}}, o_{t+T}^{\text{KG}}, l\rangle$, and the transition for the low level decision making will be formulated as $\langle (o_t^{\text{KG}}, g), a_t, r_{t+1}^{\text{sub}}, r_{t+1}^{\text{goal}}, (o_{t+1}^{\text{KG}}, g), l\rangle$. For every $F_{\text{up}}^{\text{meta}}$ ($F_{\text{up}}^{\text{sub}}$) time steps, we sample a batch of transitions from the corresponding replay buffer B^{meta} (B^{sub}) to update the policy π^{meta} (π^{sub}). We adopt two additional training strategies, whose effectiveness have been empirically validated by previous work [2]. Firstly, for the transitions collected within an episode, instead of directly adding them to the replay buffer, we only store them if the agent works well enough in this episode. We compute the average reward among the new

Algorithm 1 The Training Process

Input: game sets $\{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}\}$, replay buffers $\{B^{\text{meta}}, B^{\text{sub}}\}$, update frequencies $\{F_{\text{up}}^{\text{meta}}, F_{\text{up}}^{\text{sub}}\}$, validation frequency F_{val}, threshold P coefficients β, λ, τ,
Initialize: counters $N_{\text{acc}} \leftarrow \emptyset$, $N_{\text{epi}} \leftarrow \emptyset$, $k \leftarrow 1$, $p \leftarrow 0$, best validation score $V_{\text{val}} \leftarrow 0$, $r^{\text{meta}} \leftarrow 0$, caches $\{C^{\text{meta}}, C^{\text{sub}}\}$, policies $\{\pi^{\text{meta}}, \pi^{\text{sub}}\}$, $\{\Pi^{\text{meta}}, \Pi^{\text{sub}}\}$

1: **for** $e \leftarrow 1$ to NUM_EPISODES **do**
2: $l \leftarrow \text{SampleLevel}(\mathcal{L}, p_l)$
3: $x \leftarrow \text{SampleGame}(\mathcal{D}_{\text{train}}, l)$
4: $o_0^{\text{KG}} \leftarrow$ reset x
5: $C^{\text{meta}} \leftarrow \emptyset$, $C^{\text{sub}} \leftarrow \emptyset$, $N_{\text{epi}} \leftarrow \emptyset$,
6: Update N_{acc}, N_{epi} with o_0^{KG}
7: **for** $t \leftarrow 0$ to NUM_STEPS **do**
8: $g \leftarrow \pi^{\text{meta}}(g|o_t^{\text{KG}})$
9: $r^{\text{meta}} \leftarrow 0$
10: **while** g is not terminated **do**
11: $a_t \leftarrow \pi^{\text{sub}}(a|o_t^{\text{KG}}, g)$
12: Execute a_t, receive o_{t+1}^{KG}, r_{t+1}^{env}, obtain r_{t+1}^{goal}
13: Update N_{acc}, N_{epi} with o_{t+1}^{KG}
14: Compute r_{t+1}^{sub} using Eq. (6.2) and Eq. (6.3)
15: Push the low level transitions into C^{sub}
16: $r^{\text{meta}} \leftarrow r^{\text{meta}} + r_{t+1}^{\text{env}}$
17: $t \leftarrow t + 1$
18: $k \leftarrow k + 1$
19: **if** $k \% F_{\text{up}}^{\text{meta}} = 0$ **then**
20: $\text{Update}(\pi^{\text{meta}}, B^{\text{meta}})$
21: **if** $k \% F_{\text{up}}^{\text{sub}} = 0$ **then**
22: $\text{Update}(\pi^{\text{sub}}, B^{\text{sub}})$
23: Push the high level transitions into C^{meta}
24: Update p_l using Eq. (6.4)
25: **if** $\text{Avg}(r^{\text{meta}}|C^{\text{meta}}, l) > \tau \cdot \text{Avg}(r^{\text{meta}}|B^{\text{meta}}, l)$ **then**
26: Store all high level transitions in C^{meta} into B^{meta}
27: **if** $\text{Avg}(r^{\text{goal}}|C^{\text{sub}}, l) > \tau \cdot \text{Avg}(r^{\text{goal}}|B^{\text{sub}}, l)$ **then**
28: Store all low level transitions in C^{sub} into B^{sub}
29: **if** $e \% F_{\text{val}} = 0$ **then**
30: $v_{\text{val}} \leftarrow \text{Validate}(\pi^{\text{meta}}, \pi^{\text{sub}}, \mathcal{D}_{\text{val}})$
31: **if** $v_{\text{val}} \geq V_{\text{val}}$ **then**
32: $V_{\text{val}} \leftarrow v_{\text{val}}$, $\Pi^{\text{meta}} \leftarrow \pi^{\text{meta}}$, $\Pi^{\text{sub}} \leftarrow \pi^{\text{sub}}$
33: $p \leftarrow 0$, continue
34: **if** $p > P$ **then**
35: $\pi^{\text{meta}} \leftarrow \Pi^{\text{meta}}$, $\pi^{\text{sub}} \leftarrow \Pi^{\text{sub}}$, $p \leftarrow 0$
36: **else**
37: $p \leftarrow p + 1$

transitions, and compare it with the average reward of the stored transitions times a coefficient τ (lines 25-28). Secondly, for every F_{val} episodes, we validate the agent on a hold-out validation set $\mathcal{D}_{\text{val}}$. We track the best validation score V, as well as the corresponding policies Π^{meta} and Π^{sub}. A threshold is set that if the agent exhibits inferior performance (i.e., $v_{\text{val}} < V$) for over P validation times, the current policies $\{\pi^{\text{meta}}, \pi^{\text{sub}}\}$ will be restored as $\{\Pi^{\text{meta}}, \Pi^{\text{sub}}\}$ (lines 29-37).

If we regard a "task" as learning to solve those from a specific difficulty level, the setting of training on those from a mixture of difficulty levels can be treated as Multi-task Learning (MTL). Although these levels share some knowledge (e.g., overall objective, vocabulary set), different amount of effort should be made in learning. For example, the "hard" games typically require more interaction data and training time, and the agent tends to perform badly on them. In order to make the agent be capable of such MTL setting, we enhance Algorithm 1 with two training techniques: 1) the scheduled task sampling strategy, and 2) the level-aware replay buffers. Regarding the first technique, we draw inspiration from the curriculum learning [29] to arrange the tasks by their difficulties. For each level l, we keep track of the agent's training performance upon it v_l. Then we compute the probability for sampling a level through softmax-like operation:

$$p_l = \frac{e^{\beta - v_l}}{\sum_{l_i \in \mathcal{L}} e^{\beta - v_{l_i}}} \tag{6.4}$$

where the coefficient β is a constant. Before an episode, instead of sampling a game regardless of its level, we will first select a level based on p_l, then uniformly select a game belonging to this level (lines 2-3). In this way, we can encourage the agent to pay more attention to those hard games as the learning goes on. We conduct another technique, the level-aware replay buffers, during the process of storing newly collected transitions into the replay buffer. In hard games, the agent tends to perform badly, making the transitions less likely to be pushed into the replay buffer. We use the level-aware replay buffers to solve this problem. Specifically, we split the replay buffer by the game levels, then, the new transitions will be compared and stored to the "sub replay buffer" corresponding to their level (lines 25-28).

6.5 Experiments

6.5.1 Experiment Setting

The experiments are conducted upon cooking games with a range of multiple levels[2] [53]. Adhikari et al. [2] also studied the language-conditional RL in the cooking domain, while we extend their experiment setting from 4 levels to 8 levels, and consider more difficult generalization type -

[2] `https://aka.ms/twkg/rl.0.1.zip`

TABLE 6.1: The game statistics.

Level	#Triplets	#Rooms	#Objs	#Ings	#Reqs	#Acts	MaxScore
S1	21.44	1	17.09	1	1	11.54	4
S2	21.50	1	17.49	1	2	11.81	5
S3	46.09	9	34.15	1	0	7.25	3
S4	54.54	6	33.41	3	2	28.38	11
US1	19.85	1	16.01	1	0	7.98	3
US2	20.74	1	16.69	1	1	8.87	4
US3	33.04	6	24.81	1	0	7.61	3
US4	47.31	6	31.09	3	0	13.90	5

generalization upon unseen levels. Our training set $\mathcal{D}_{\text{train}}$ contains a mixture of 4 levels, where for each level there are 100 games. Our validating set $\mathcal{D}_{\text{val}}$ consists of the same levels with $\mathcal{D}_{\text{train}}$, where for each level there are 20 games. We construct two testing sets, $\mathcal{D}_{\text{test}}^{\text{seen}}$, and $\mathcal{D}_{\text{test}}^{\text{unseen}}$. Both testing sets have 4 levels, and each level contains 20 games. While $\mathcal{D}_{\text{test}}^{\text{seen}}$ share same mixture of levels with $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$, the game levels in $\mathcal{D}_{\text{test}}^{\text{unseen}}$ are not encountered during training. Table 6.1 displays the statistics of each level, where a seen level is denoted as "S#" ($\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, and $\mathcal{D}_{\text{test}}^{\text{seen}}$), and an unseen level is denoted as "US#" ($\mathcal{D}_{\text{test}}^{\text{unseen}}$). "#Ings", "#Reqs", "#Acts" denote the average number of the ingredients, the requirements, and the current admissible actions, respectively. More examples about the games & levels are provided in Appendix A.

6.5.2 Baselines

Our agent and the baselines are shown as follows:

- GATA [2], which is the state-of-the-art agent in the cooking domain. This agent does not contain the hierarchical architecture. Besides, during the action selection process, neither the goal nor the subtask is provided to the agent.
- GC-GATA, which extends GATA with a **g**oal-**c**onditioned policy for action selection. However, it does not have a meta-policy that the goal is uniformly sampled from the currently available goal set.
- H-KGA, which is the proposed agent.

- H-KGA HalfJoint, which is a variant of H-KGA. In the first **half** of the training process, we only train the sub-policy. Then in the second half, we train both the sub-policy policy and the meta-policy **joint**ly .
- H-KGA Ind, which is another variant of H-KGA with **ind**ividually trained policies. In other words, we only train the sub-policy in the first half. Then we freeze the sub-policy, and train the meta-policy in the second half.

6.5.3 Implementation and Training Details

We treat GATA as our building backbone and implement H-KGA upon it. In particular, we adopt the version GATA-GTF, where the full knowledge graph is provided as the observation, and the textual observation is discarded, thus eliminating the influence of error incurred during information extraction. For all agents, we implement the graph encoder using R-GCNs, implement the text encoder using a single head transformer block, and implement the scorers using fully-connected layers. For H-KGA, we initialize the encoders in both policies as separate modules. We implement a non-learnable goal set generator for obtaining the goal set $\mathcal{G}_t$, where Algorithm 2 shows the pipeline. We first obtain the ingredient set $\mathcal{I}$. For each ingredient $i \in \mathcal{I}$, we first check whether it has been collected, then obtain its status set $\mathcal{S}_i$ and requirement set $\mathcal{R}_i$. We consider three types of goals: 1) "find" requires the agent to find and collect an uncollected ingredient, 2) "prepare" requires the agent to prepare an ingredient to satisfy a requirement, and 3) "eat", that the agent is required to prepare and eat the final meal. Algorithm 3 shows the pipeline for assigning the goal-conditioned reward r_t^{goal}. We first obtain the type of a goal g, then check whether this goal has been accomplished given a_t and o_{t+1}^{KG}. Some functions in Algorithm 2 can be reused here. r_t^{goal} is a binary reward that we will assign $r_t^{\text{goal}} = r_{\text{max}}$ if g is accomplished successfully, otherwise r_{min} (still not finished, or failed).

We define a training episode as 50 steps, and a validation / testing episode as 100 steps. All agents are trained for 100,000 episodes. The BeBold method, the scheduled task sampling strategy and the level-aware replay buffer are applied among all models, where the constant coefficient λ is set as 0.1, and β is set as 1.0. We set the sizes of the replay buffers B^{meta} and B^{sub} as 50,000 and 500,000, respectively. For every $F_{\text{up}}^{\text{meta}} = F_{\text{up}}^{\text{sub}} = 50$ steps, we optimize the policies with batch size 64. We initialize the two H-KGA variants, H-KGA HalfJoint and H-KGA Ind, with the GC-GATA agent

Algorithm 2 Goal set generation

Input: Knowledge graph o_t^{KG}
Initialize: Goal set $\mathcal{G}_t \leftarrow \emptyset$

Get ingredient set $\mathcal{I} \leftarrow \text{GetIng}(o_t^{\text{KG}})$
for each ingredient $i \in \mathcal{I}$ **do**
    if CheckCollection$(i, o_t^{\text{KG}}) =$ False **then**
        Add goal "find i" to $\mathcal{G}_t$
    else
        Get status set $\mathcal{S}_i \leftarrow \text{GetStatus}(i, o_t^{\text{KG}})$
        Get requirement set $\mathcal{R}_i \leftarrow \text{GetReq}(i, o_t^{\text{KG}})$
        for each requirement $r_i \in \mathcal{R}_i$ **do**
            if $r_i \notin \mathcal{S}_i$ **then**
                Add goal "r_i i" to $\mathcal{G}_t$
if $\mathcal{G}_t = \emptyset$ **then**
    Add goal "prepare and eat meal" to $\mathcal{G}_t$
return $\mathcal{G}_t$.

Algorithm 3 Goal-conditioned reward acquisition

Input: Knowledge graph o_{t+1}^{KG}, goal g, rewards $\{r_{\text{min}}, r_{\text{max}}\}$
Initialize: FLAG $\leftarrow$ False

Obtain goal type $g_{\text{type}} \leftarrow \text{GetGoalType}(g)$
if $g_{\text{type}} =$ "find" **then**
    Get ingredient i from g
    if CheckCollection$(i, o_{t+1}^{\text{KG}}) =$ True **then**
        FLAG $\leftarrow$ True
else if $g_{\text{type}} =$ "prepare" **then**
    Get ingredient i, requirement r from g
    Get status set $\mathcal{S}_i \leftarrow \text{GetStatus}(i, o_{t+1}^{\text{KG}})$
    if $r \in \mathcal{S}_i$ **then**
        FLAG $\leftarrow$ True
else
    if CheckExistance("meal", o_{t+1}^{KG}) = True **then**
        FLAG $\leftarrow$ True
if FLAG = True **then**
    return r_{max}.
else
    return r_{min}.

being pre-trained for 50,000 episodes. For every $F_{\text{val}} = 1000$ episodes, we conduct validation on $\mathcal{D}_{\text{val}}$, and test the agents on $\mathcal{D}_{\text{test}}^{\text{seen}}$ and $\mathcal{D}_{\text{test}}^{\text{unseen}}$.

6.5.4 Evaluation Metrics

We define the score of a game as the sum of rewards collected in an episode. To make the results of different levels comparable, we scale the score to $[0, 1]$ by dividing it with the maximum score of

TABLE 6.2: The testing result at the end of the training process.

Model	S1	S2	S3	S4	US1	US2	US3	US4	Avg Seen	Avg Unseen	Avg All
GATA	0.70±0.08	0.43±0.02	0.48±0.04	0.28±0.05	0.90±0.04	0.59±0.08	0.71±0.11	0.28±0.09	0.47±0.04	0.62±0.07	0.55±0.06
GC-GATA	0.68±0.18	0.73±0.22	0.43±0.04	0.33±0.23	0.78±0.15	0.79±0.12	0.59±0.11	0.28±0.14	0.54±0.13	0.61±0.12	0.58±0.12
H-KGA (ours)	**0.93**±0.06	0.85±0.17	0.50±0.05	**0.61**±0.18	0.97±0.03	**0.97**±0.03	**0.78**±0.03	**0.44**±0.12	**0.72**±0.04	**0.79**±0.04	**0.76**±0.03
H-KGA HalfJoint	0.80±0.12	0.75±0.12	0.43±0.07	0.28±0.11	0.82±0.02	0.81±0.08	0.48±0.16	0.19±0.03	0.56±0.11	0.57±0.07	0.57±0.09
H-KGA Ind	0.91±0.00	**0.92**±0.04	0.49±0.02	0.47±0.04	0.95±0.07	0.88±0.00	0.64±0.04	0.27±0.03	0.70±0.02	0.68±0.01	0.69±0.02
GATA w/o BeBold	0.78±0.11	0.62±0.13	0.50±0.12	0.24±0.08	**0.99**±0.02	0.65±0.06	0.73±0.10	0.36±0.07	0.54±0.06	0.68±0.02	0.61±0.03
H-KGA w/o BeBold	0.73±0.14	0.74±0.07	0.43±0.11	0.37±0.09	0.90±0.14	0.77±0.01	0.64±0.10	0.31±0.16	0.57±0.07	0.65±0.09	0.61±0.07
H-KGA w/o Sch	0.73±0.19	0.60±0.16	0.39±0.08	0.34±0.04	0.91±0.01	0.80±0.13	0.52±0.09	0.29±0.07	0.52±0.07	0.63±0.07	0.57±0.05
H-KGA w/o Sch w/o LR	0.85±0.19	0.81±0.14	**0.54**±0.10	0.33±0.15	0.82±0.23	0.79±0.17	0.60±0.16	0.30±0.10	0.63±0.14	0.63±0.16	0.63±0.15

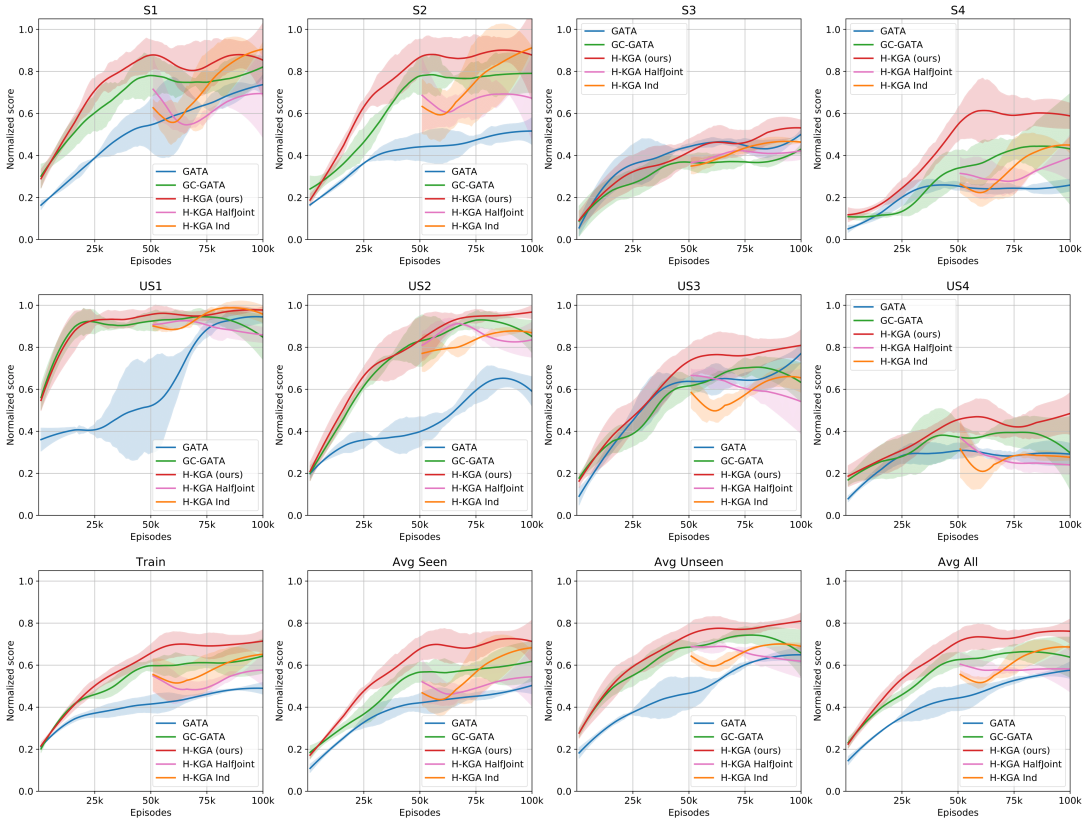


FIGURE 6.2: The performance of agents on $\mathcal{D}_{\text{test}}^{\text{seen}}$ ("S4", "Avg Seen") and $\mathcal{D}_{\text{test}}^{\text{unseen}}$ ("US4", "Avg Unseen").

this level (as shown in Table 6.1) Besides reporting the average performance of each testing set, we also show the testing result for each single level.

6.6 Results and Discussions

6.6.1 Main Results

Fig. 6.2 displays the agents' testing performance during training, and Table 6.2 compares the final testing result when training is done. The proposed agent, H-KGA, achieves better performance than the baselines in games from both seen and unseen levels. In particular, it significantly outperforms other agents in "S4", which is the most complex level in Table 4.1 (most ingredients, preparation requirements, and rooms). Two aspects, the adaptive goal selection achieved by the meta-policy, and the goal-conditioned action selection achieved by the sub-policy, collectively contribute to H-KGA's performance improvement. The effectiveness of the goal-conditioned learning can also be observed from the GC-GATA baseline, which could be treated as the sub-policy of H-KGA. While GC-GATA also improves the performance over the backbone GATA, H-KGA achieves a larger performance gain through using a learned meta-policy. We then consider another two baselines, H-KGA HalfJoint and H-KGA Ind, to compare the joint learning method we use with the individual training [105]. The H-KGA HalfJoint baseline, where the policies are jointly trained after pre-training the sub-policy in the first half, encounters performance drop, which might be caused by the RL forgetting problem [200]. Regarding the H-KGA Ind, where the meta-policy and the sub-policy are trained individually, achieves better performance than GC-GATA, but is still surpassed by the proposed H-KGA. We observe that there's still space for the H-KGA Ind baseline to improve its performance. However, it requires us to train H-KGA Ind for more episodes to collect enough interaction data, which is undesirable for the real life applications. In contrast, our H-KGA is more data efficient, that it is able to obtain comparable result with much fewer training episodes.

We also find that being equipped with a learnable meta-policy is helpful to improve generalizability for those from unseen levels. For example, in order to solve the games from "US4" more efficiently, it requires the agent to determine the collecting order of the ingredients, since there are multiple ingredients located in multiple rooms. Compared with the baselines without the meta-policy (i.e., GATA and GC-GATA), and the baselines whose meta-policy is "not-well-trained" (i.e., H-KGA Ind and H-KGA HalfJoint), our agent deals better with these games.

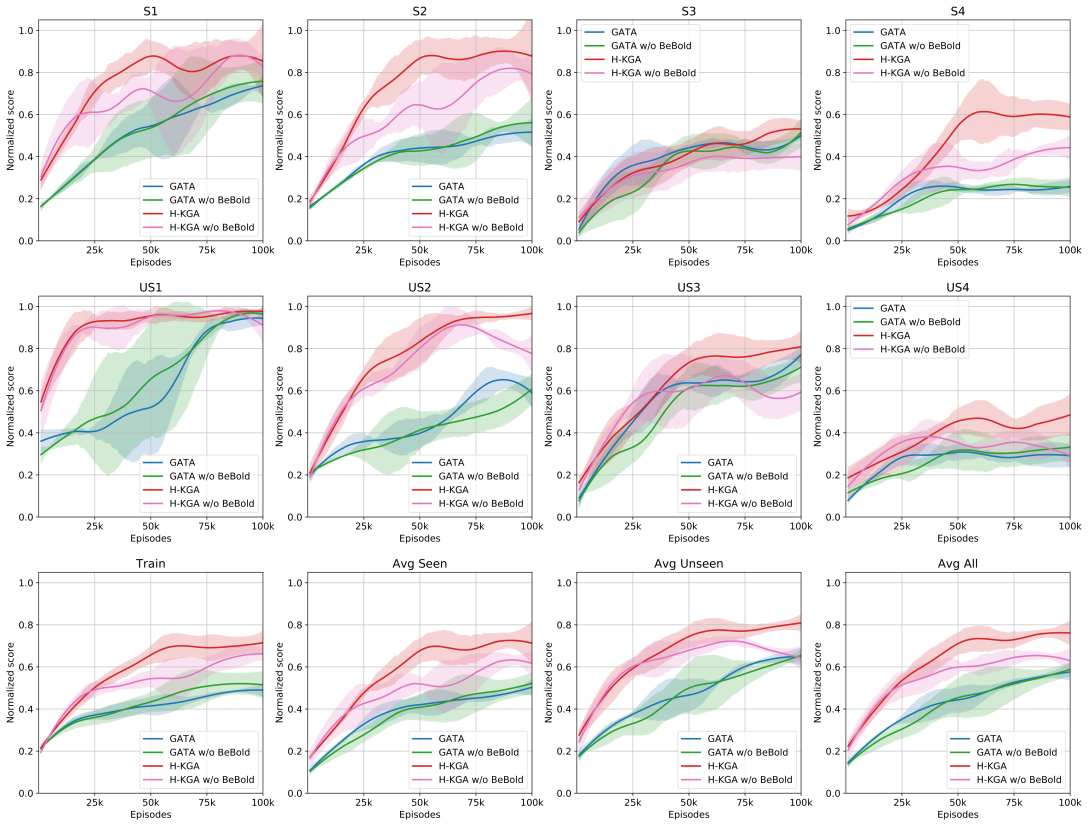


FIGURE 6.3: The performance of agents with or without the BeBold method.

6.6.2 The Influence of the BeBold Method

As discussed in Sec. 6.4.3, with the aim of conducting more effective exploration, we introduce the BeBold method to the sub-policy. In this section, we compare the H-KGA variants with or without this method to study its contribution. As shown in Fig. 6.3, even without using the BeBold method, the proposed H-KGA already significantly outperforms GATA in some simple levels (by comparing "H-KGA w/o BeBold" with "GATA w/o BeBold"). The agent still requires sufficient exploration to solve games from some difficult levels. For example, the H-KGA agent without the BeBold method can hardly reach 0.5 in "S4", and reach 0.4 in "US4" (by comparing "H-KGA" with "H-KGA w/o BeBold"). It can also be observed that only encouraging exploration is not adequate to improve the agent, as GATA benefits much less from the BeBold method (by comparing "GATA" with "GATA w/o BeBold").

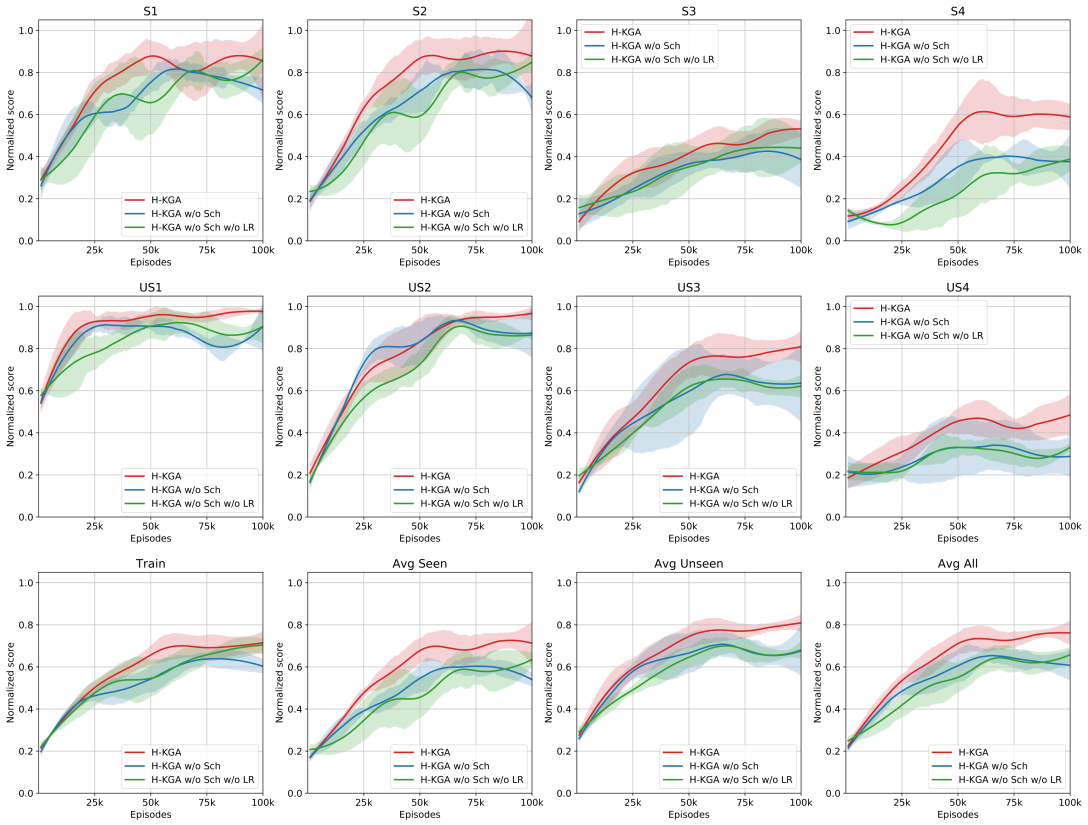


FIGURE 6.4: The performance of agents with or without the scheduled task sampling strategy (Sch) / level-aware replay buffer (LR).

6.6.3 The Influence of the MTL Training Techniques

As discussed in Sec. 6.4.4, two techniques are introduced to facilitate the MTL training settings. In this section, we compare the H-KGA variants with or without these techniques to study their contributions. We consider two variants, "H-KGA w/o Sch", where the **sch**eduled task sampling strategy is removed, and "H-KGA w/o Sch w/o LR", where both the sampling strategy and the **l**evel-aware **r**eplay buffer are removed. As shown in Fig. 6.4, without the scheduled task sampling strategy, H-KGA is still comparable in terms of the average performance ("H-KGA w/o Sch"). However, the performance of this variant is limited in games from more difficult levels, where more training samples are required. Similarly, without using the level-aware replay buffer, the transitions belonging to the difficult levels are less likely to be stored in the replay buffer, resulting in low data efficiency.

6.7 Conclusion

In this chapter, we investigated generalization for language-conditional RL from the perspective of solving text-based games. We introduced a two-level hierarchical framework, H-KGA, to tackle the generalization problem. In the high level, we considered a meta-policy for task decomposition and subtask selection. Then, in the low level, we considered a sub-policy for subtask-conditioned action selection. Besides, we proposed two techniques, the scheduled task sampling strategy, and the level-aware replay buffers, to facilitate training under the multi-task learning setting. We empirically validated the effectiveness of H-KGA upon 8 game sets with different levels and generalization types.

With respect to the limitations & future directions, the H-KGA model still uses heuristic-based goal set generation and goal-condition reward acquisition methods. As a future direction, we would like to study automatic methods to achieve such functionalities. Besides, we assume the accessibility of full knowledge graph as the observation, which does not hold in practice. We would like to alleviate such assumption in the future work. As this work is conducted under the setting of static training / testing game sets, we are also interested in extending our work to more difficult and realistic RL settings, such as the procedurally-generated environment, where one game instance can only be interacted for one episode.

Chapter 7

Question-guided World-perceiving Agent

In this chapter, we study the challenges of low sample efficiency and large action space for language-conditional RL. We address these challenges by introducing the world-perceiving modules, which automatically decompose tasks and prune actions through answering questions about the environment. We then propose a two-phase training framework to decouple language learning from reinforcement learning, which further improves the sample efficiency. The experimental results indicate that our method improves the sample efficiency and the performance by a large margin. Besides, it shows robustness against compound error and limited pre-training data.

7.1 Introduction

Despite the effectiveness and premises, two major challenges prevent the RL agents from being deployed in real world applications: the *low sample efficiency*, and the *large action space* [63]. The low sample efficiency is a crucial limitation of RL, which refers to the fact that it typically requires massive training data to obtain a human-level agent [195]. Human beings are usually armed with prior knowledge, so that they don't have to learn from scratch [61]. In a language-conditional RL system, in contrast, the agent is required to conduct both language learning and decision making regimes, where the former can be considered as prior knowledge and is much slower than the

later [93]. The sample efficiency could be improved through pre-training methods, which decouple the language learning from decision making [188]. The selection of pre-training methods thus plays an important role: if the pre-trained modules perform poorly on unseen data during RL training, the incurred compound error will severely affect the decision making process. Another challenge is the large discrete action space: the agent may waste both time and training data if attempting irrelevant or inferior actions [62, 227].

Our aim in this chapter is to provide solutions to these two challenges. Since it might be inefficient to train an agent to solve complicated tasks (games) from scratch, we consider decomposing a task into a sequence of subtasks as inspired by [12]. We design an RL agent that is capable of automatic task decomposition and subtask-conditioned action pruning, which brings two branches of benefits. First, the subtasks are easier to solve, as the involved temporal dependencies are usually short-term. Second, by acquiring the skills to solve subtasks, the agent can be adapted to a new task more quickly by reusing the learnt skills [22]. The challenge of large action space can also be alleviated, if we can filter out the actions that are irrelevant to the current subtask.

Inspired by the observation that human beings understand the environment conditions through question answering [10, 55], we design world-perceiving modules to realize the aforementioned functionalities (i.e., task decomposition and action pruning) and name our method as **Q**uestion-guided **W**orld-perceiving **A**gent (QWA). Fig. 7.1 (b) shows an example of our decision making process. Being guided by some questions, the agent first decomposes the task to get a set of available subtasks, and selects one from them. Next, conditioned on the selected subtask, the agent conducts action pruning to obtain a refined set of actions. In order to decouple language learning from decision making, which further improves the sample efficiency, we propose to acquire the world-perceiving modules through supervised pre-training. We design a two-phase framework to train our agent. In the first phase, a dataset is built for the training of the world-perceiving modules. In the second phase, we deploy the agent in games with the pre-trained modules frozen, and train the agent through reinforcement learning.

We evaluate our agent and training framework on a range of cooking games. We divide the games as simple games and complex games, and construct the pre-training dataset from simple games only. The experimental results show that QWA achieves high sample efficiency in solving complex games.

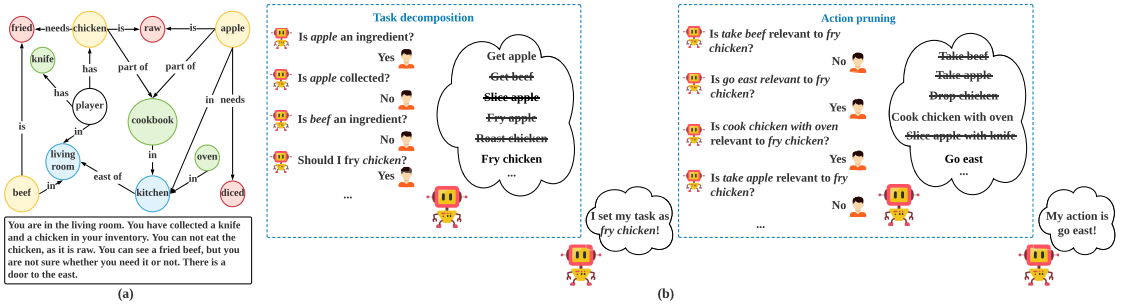


FIGURE 7.1: (a) An example of the observation, which can be textual, KG-based, or hybrid. (b) The decision making process. Through question answering, the agent is guided to first decompose the task as subtasks, then reduce the action space conditioned on the subtask.

We also show that our method enjoys robustness against compound error and limited pre-training data.

Our contributions are summarized as following three aspects:

1. We develop an RL agent featured with question-guided task decomposition and action space reduction.
2. We design a two-phase framework to efficiently train the agent with limited data.
3. We empirically validate our method's effectiveness and robustness in complex games.

7.2 Related Work

In this work, we concentrate on improving the sample efficiency and reducing the action space through pre-training. Being agnostic about the form of observations and the action selecting methods, our work complements the existing RL agents.

7.2.1 Hierarchical RL

Our work is closely associated with the task decomposition [157, 179, 184] and hierarchical reinforcement learning [57, 120, 199]. Similar efforts have been made by Jiang et al. [105] and Xu et al. [215], who designed a meta-policy for task decomposition and subtask selection, and a sub-policy for goal-conditioned decision making. Typically, these works either assume the

access to a set of available subtasks, or decompose a task through pre-defined rules, while we aim to achieve automatic task decomposition through pre-training, and remove the requirement for expert knowledge during reinforcement learning. Besides, existing work assumes that unlimited interaction data can be obtained to train the whole model through RL. In contrast, we consider the more practical situation where the interaction data is limited, and focus on improving the RL agent's data efficiency. Regarding the sub-policy, we do not make the assumption about the accessibility of the termination states. We also do not require additional handcrafted operations in reward shaping [21].

7.2.2 Pre-training Methods for RL

There have been a wide range of work studying pre-training methods or incorporating pre-trained modules to facilitate reinforcement learning [65, 75, 83, 130, 177, 178]. One major branch among them is the Imitation Learning (IL), in which the agent is trained to mimic the human demonstrations before being deployed in RL [92, 167, 240]. Although we also collect human labeled data for pre-training, we utilize the data to aid the agent to perceive the world, instead of learning the solving strategies. Therefore, we do not require the demonstrations to be perfect to solve the game. Besides, our method prevails when pre-trained on simple tasks rather than complicated ones, which is more feasible for human to interact and annotate [19, 140]. Further discussions to compare our method with IL are provided in subsequent sections.

In the domain of text-based games, some prior works have involved pre-training tasks such as state representation learning [11, 183], knowledge graph constructing [147] and action pruning [86, 193, 219]. For instance, Ammanabrolu et al. [10] designed a module to extract triplets from the textual observation by answering questions, and use these triplets to update the knowledge graph. As far as we know, we are the first to incorporate pre-training based task decompositon in this domain. Besides, instead of directly pruning the actions based on the observation, we introduce subtask-conditioned action pruning to further reduce the action space.

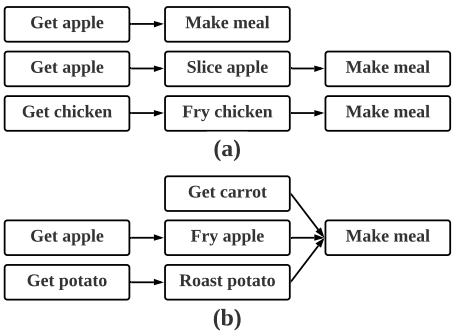


FIGURE 7.2: Subtasks for solving (a) 3 simple games and (b) 1 complex game.

7.3 Problem Statement

As illustrated in previous chapters, the observation could be textual, knowledge graph-based, or hybrid. Fig. 7.1 (a) gives an example of the textual observation, and its corresponding KG-based observation. We do not make assumptions about the observation form and our method is compatible with any of those forms.

Our objective is to make an agent capable of automatic task decomposition and action pruning in solving text-based games. we only consider games having similar themes, but are different in their complexities [2, 49]. Taking the cooking games [53] as an example, the task is always "make the meal". To accomplish this task, the agent has to explore different rooms to pick up all ingredients, cook them with right methods, and make the meal. A game's complexity depends on the number of rooms, ingredients, and the required preparation steps. We define a subtask as a milestone towards completing the task (e.g., "get apple" if "apple" is included in the recipe), and a subtask requires a chain of multiple actions to accomplish (e.g., exploring the house to pick up the apple). A game is considered simple, if it consists of only a few subtasks, and complex if it consists of more subtasks. Fig. 7.2 gives examples of simple games and complex games. While being closer to real world applications, complex games are hard to solve by RL agents because: 1) it's expensive to collect sufficient human labeled data for pre-training; 2) it's unrealistic to train an RL agent from scratch. We therefore focus on agent's sample efficiency and performance on complex games. Our objective is to leverage the labeled data collected from simple games to speed up RL training in complex games, thus obtaining an agent capable of complex games.

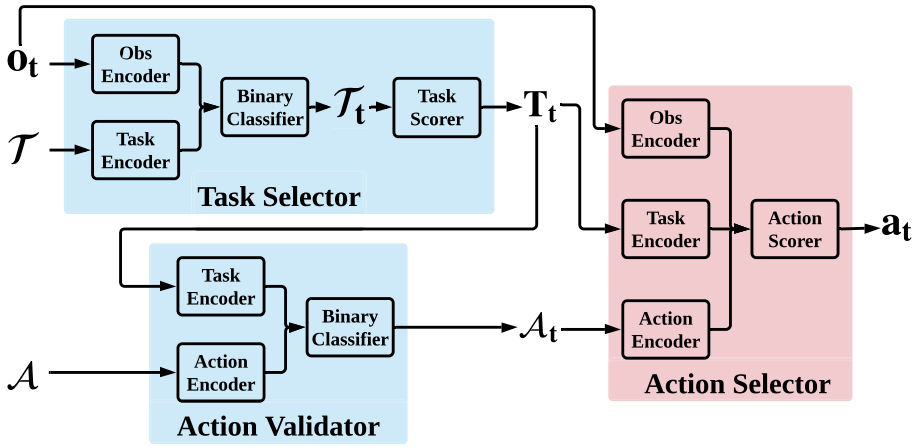


FIGURE 7.3: The overview of QWA. The blue modules will be trained in the pre-training phase, while the red module will be trained in the RL phase.

7.4 Methodology

7.4.1 Framework Overview

Fig. 7.3 presents our QWA agent. We consider two world-perceiving modules: a task selector and an action validator. Given the observation o_t and the task candidate set $\mathcal{T}$, we use the task selector to first get a subset of currently available subtasks $\mathcal{T}_t \subseteq \mathcal{T}$, then choose a subtask $T_t \in \mathcal{T}_t$. Given T_t and the action candidate set $\mathcal{A}$, we use the action validator to get an action subset $\mathcal{A}_t \subseteq \mathcal{A}$, which contains only those relevant to the subtask T_t. Finally, given o_t and T_t, we use an action selector to score each action $a \in \mathcal{A}_t$, then select the action with the largest score as a_t.

The training of world-perceiving modules can be regarded as the language learning regime, while the training of the action selector can be regarded as the decision making regime. We consider a two-phase training strategy to decouple these two regimes to further improve the sample efficiency [93]. In the pre-training phase, we collect human interaction data from those simple games, and design QA datasets to train the world-perceiving modules through supervised learning. In the reinforcement learning phase, we freeze the pre-trained modules, and train the action selector in the complex games through reinforcement learning.

7.4.2 Task Selector

Depending on the experiment settings, $\mathcal{T}$ and $\mathcal{A}$ can be either fixed vocabulary sets (parser-based), or changing over time (choice-based). We regard a subtask available if it is essential for solving the "global" task, and there's no prerequisite subtask. For example, the subtask "get apple" in Fig. 7.1, as the object "apple" is an ingredient which has not been collected. Although another subtask "dice apple" is also essential for making the meal, it is not available since there exists a prerequisite subtask (i.e., you should collect the apple before dicing it). The task selector aims at identifying a subset of available subtasks $\mathcal{T}_t \subseteq \mathcal{T}$, and then select one subtask $T_t \in \mathcal{T}_t$.

We formulate the mapping $f(o_t, \mathcal{T}) \rightarrow \mathcal{T}_t$ as a multi-label learning problem [233]. For simplicity, we assume that the subtask candidates are independent with each other. Thus, the multi-label learning problem can be decomposed as $|\mathcal{T}|$ binary classification problems. Inspired by the recent progress of question-conditional probing [55], language grounding [93], and QA-based graph construction [10], we cast these binary classification problems as yes-or-no questions, making the task selector a world-perceiving module. For example, the corresponding question for the subtask candidate "get apple" could be "Whether 'get apple' is an available subtask?". This module can guide the agent to understand the environment conditions through answering questions, but will not directly lead the agent to a specific decision. We can obtain this module through supervised pre-training, and decouple it from reinforcement learning to yield better sample efficiency. Fig. 7.1 (b) shows some sample QAs, where a human answerer can be replaced by a pre-trained task selector.

Some previous work also considered task decomposition [49, 97], but the related module is obtained through imitating human demonstrations, which is directly related to decision making instead of world perceiving. Compared with these work, our method has two folds of benefits. First, there may exist multiple available subtasks at a timestep. Imitating human demonstrations will specify only one of them, which may be insufficient and lead to information loss. Second, we do not require expert demonstrations which guarantee to solve the game. Instead, we can ask humans to annotate either imperfect demonstrations, or even demonstrations from a random agent. We will treat the IL-based method as a baseline and conduct comparisons in the experiments.

Given the set of available subtasks $\mathcal{T}_t$, alternative strategies can be used to select a subtask T_t from it. For example, we can employ a non-learnable task scorer to obtain T_t by random sampling, since each subtask $T \in \mathcal{T}_t$ is essential for accomplishing the task. We can also train a task scorer via a meta-policy for adaptive task selection [215].

7.4.3 Action Validator

After obtaining the subtask T_t, we conduct action pruning conditioned on it (or on both T_t and o_t) to tackle the challenge of the large action set. Similar to the task selector, we formulate action pruning as $|\mathcal{A}|$ binary classification problems, and devise another world-perceiving module: the action validator. The action validator is designed to check the relevance of each action candidate $a \in \mathcal{A}$ with respect to T_t by answering questions like "Is the action candidate 'take beef' relevant to the subtask 'fry chicken'?", so as to obtain a subset of actions $\mathcal{A}_t \subseteq \mathcal{A}$ with irrelevant actions filtered. Fig. 7.3 shows the module architecture. Similar to the task selector, we pre-train this module through question answering. Sample QAs have been shown in Fig. 7.1 (b).

7.4.4 Action Selector

After pre-training, we deploy the agent in the complex games, and train the action selector through RL. We freeze the pre-trained modules, as in this phase no human labeled data will be obtained. At each time step, we use the task selector and the action validator to produce $\mathcal{T}_t$ and $\mathcal{A}_t$, respectively. We keep using the same subtask T over time until it is not included in $\mathcal{T}_t$, as we do not want the agent to switch subtasks too frequently. The agent can simply treat T_t as the additional observation of o_t. If we do not limit the use of human knowledge in this phase, we can also treat T_t as a goal with either hand-crafted [105] or learnt reward function [52]. Arbitrary methods can be used for optimizing [2, 7].

One issue we are concerned about is the compound error $-$ the prediction error from imperfect pre-trained modules will adversely affect RL training [163, 192]. For example, the false predictions made by the binary classifier in the task selector may lead to a wrong T_t, which affects $\mathcal{A}_t$ and a_t in turn. To alleviate the influence of the compound error, we assign time-awareness to subtasks. A subtask is bounded by a time limit $[0, \xi]$. If the current subtask T is not finished within its time limit,

TABLE 7.1: Game statistics. We use the simple games to provide human labeled data during pre-training, and use the medium & hard games during reinforcement learning.

Name	Traj.Length	#Triplets	#Rooms	#Objs	#Ings	#Reqs	#Acts	#Subtasks	#Avail.Subtasks
Simple	7.90	38.48	5.76	23.69	1.49	0.96	14.50	12.44	1.14
Medium	15.30	51.07	6.00	26.10	3.00	3.00	23.48	23.00	1.94
Hard	21.75	59.95	8.00	31.48	3.00	4.00	22.94	23.00	2.16

we force the agent to re-select a new subtask $T_t \in \mathcal{T}_t \setminus \{T\}$, regardless whether T is still available. Besides making the agent robust against errors, another benefit by introducing time-awareness to subtasks is that it improves the subtask selection diversity, which helps the agent to avoid getting stuck in local minima [40, 162].

7.5 Experiments

7.5.1 Experiment Settings

We conduct experiments on cooking games provided by the rl.0.2 game set[1] and the FTWP game set[2]. Based on the number of subtasks, which is highly correlated to the number of ingredients & preparing requirements, we design three game sets with varying complexities: 3488 simple games, 280 medium games and 420 hard games. Note that there is no overlapping games between the simple set and the medium / hard game sets. Following [2], we simplify the game environment by making the action set changeable over time, which can be provided by the TextWorld platform [53]. Note that although the action space is reduced, it still remains challenging as the agent may encounter unseen action candidates [42, 43]. We then use a similar way to obtain a changeable task set, which is a combination of the verb set {chop, dice, slice, fry, get, grill, roast, get, make} and the ingredient set. We still denote the subtask candidate set (action candidate set) as $\mathcal{T}$ ($\mathcal{A}$) to distinguish it from the available subtask set $\mathcal{T}_t$ (refined action set $\mathcal{A}_t$). Table 7.1 shows the game statistics. Besides "Traj.Length", which denotes the average length of expert demonstrations per game[3], other statistic metrics are averaged per time step per game (e.g., "#Subtasks" and "#Avail.Subtasks" denote the average number of subtask candidates $\mathcal{T}$, and the average number of available subtasks $\mathcal{T}_t$, respectively). We will collect human interaction data from the simple games

[1] `https://aka.ms/twkg/rl.0.2.zip`

[2] `https://aka.ms/ftwp/dataset.zip`

[3] The demonstrations of the medium & hard games are just for statistics, and will not be used for pre-training.

for pre-training. We regard both medium & hard games as complex, and will conduct reinforcement learning on these two game sets.

7.5.2 Baselines

Our model and the baselines are shown as follows:

- GATA [2], which is a powerful KG-based agent and the benchmark model for cooking games.
- IL [49], which is a hierarchical agent that also uses two training phases. In the first phase, both the task selector and the action selector are pre-trained through imitation learning. Then the action selector will be fine-tuned through reinforcement learning in the second phase.
- IL w/o FT, which is a variant of the IL baseline, where only the imitation pre-training phase is conducted, and there's no RL fine-tuning.
- QWA, which is the proposed model with world-perceiving modules.

7.5.3 Implementation Details

Model architecture All models are implemented based on GATA's released code[4]. In particular, we use the version GATA-GTF, which takes only the KG-based observation, and denote it as GATA for simplicity. The observation encoder is implemented via the Relational Graph Convolutional Networks (R-GCNs) [173] by taking into account both nodes and edges. Both the task encoder and the action encoder are implemented based on a single transformer block with single head [197] to encode short texts. The binary classifier, task scorer and action scorer are linear layers. The details about GATA as well as IL baselines could be found at Appendix B.

Pre-training data collection We train the task selector and the action validator separately, as they use different types of QAs. We ask human players to play the simple games, and answer the yes-or-no questions based on the observations. The details of the dataset construction (interaction data collection, question generation, answer annotation, etc.) could be found at Appendix B. We train the task selector with a batch size of 256, and the action validator with a batch size of 64. The

[4] https://github.com/xingdi-eric-yuan/GATA-public

TABLE 7.2: The testing performance at 20% / 100% of the reinforcement learning phase.

Model	Medium		Hard	
	20%	**100%**	**20%**	**100%**
QWA (ours)	**0.66**±0.02	**0.71**±0.04	**0.53**±0.04	**0.53**±0.02
GATA	0.31±0.02	0.57±0.18	0.25±0.02	0.48±0.01
IL	0.45±0.18	0.26±0.03	0.32±0.11	0.35±0.08
IL w/o FT	0.63±0.05	0.63±0.05	0.48±0.05	0.48±0.05

modules are trained for 10-20 epochs using Focal loss and Adam optimizer with a learning rate of 0.001.

Reinforcement learning We consider the medium game set and hard game set as different experiments. We split the medium game set into 200 training games / 40 validation games / 40 testing games, and the hard game set into 300 / 60 / 60. We follow the default setting of [2] to conduct reinforcement learning. We define a training episode as 50 steps, and a validation / testing episode as 100 steps. We set the subtask time limit $\xi = 5$. For every episode, a game is sampled from the corresponding game set for the agent to interact with. The models are optimized via Double DQN (epsilon decays from 1.0 to 0.1 in 20,000 episodes, Adam optimizer with the learning rate being set as 0.001) with Pritorized Experience Replay (replay buffer size 500,000). All models are trained for 100,000 episodes. For every 1,000 training episodes, we validate the models and report the testing performance.

7.5.4 Evaluation Metrics

We measure the models through their RL testing performance. Specifically, we adopt the normalized score defined in Chapter A as the main metric.

7.6 Results and Discussions

7.6.1 Main Results

Fig. 7.4 compares the RL testing performance with respect to the training episodes. Table 7.2 shows the testing performance after 20,000 training episodes (20%) / at the end of RL training (100%). Compared with GATA, which needs to be "trained from scratch", the proposed QWA

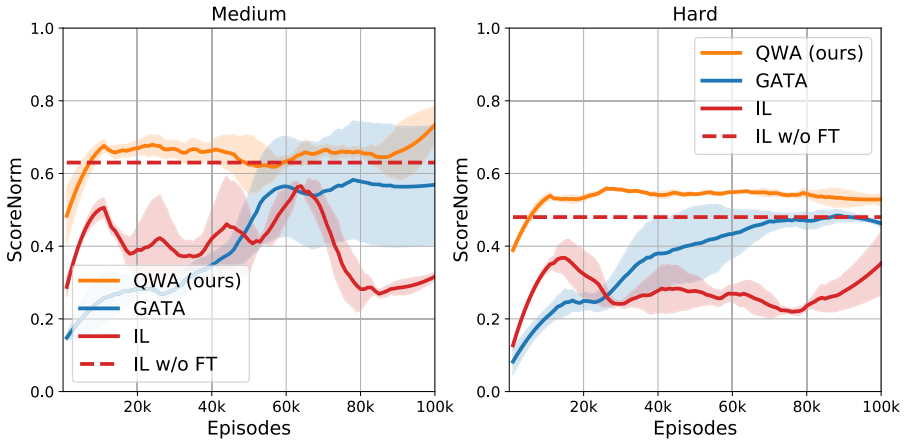


FIGURE 7.4: The RL testing performance w.r.t. training episodes. The red dashed line denotes the IL agent without fine-tuning.

model achieves high sample efficiency: it reaches convergence with high performance before 20% of the training stage, saving 80% of the online interaction data in complex games. The effectiveness of pre-training can also be observed from the variant "IL w/o FT": even though it requires no further training on the medium / hard games, it achieves comparable performance to our model. However, the performance of QWA can be further improved through RL, while it does not work for the IL-based model, as we can observe the performance of "IL" becomes unstable and drops significantly during the RL fine-tuning. A possible reason is that there exists large domain gap between simple and medium (hard) games, and our model is more robust against such domain shifts. For example, our world-perceiving task selector performs better than IL-based task selector in handling more complex observations (according to Table 7.1, the observations in medium / hard games contain more triplets, rooms and objects), facilitating the training of the action selector. Besides the domain gap in terms of the observation space, there is also a gap between domains in terms of the number of available subtasks – while there's always one available subtask per time step in simple games, the model will face more available subtasks in the medium / hard games. Different from our task selector, which is trained to check the availability of every subtask candidate, the IL pre-trained task selector can not adapt well in this situation, as it is trained to find the unique subtask and ignore the other subtask candidates despite whether they are also available.

TABLE 7.3: The RL testing performance on simple games.

Model	Medium 100%	Hard 100%
QWA (ours)	**0.80**±0.01	**0.82**±0.02
GATA	0.32±0.03	0.45±0.12
IL	0.44±0.02	0.29±0.03
IL w/o FT	0.76±0.06	0.76±0.06

7.6.2 Performance on the Simple Games

We further investigate the generalization performance of our model on simple games, considering that simple games are not engaged in our RL training. To conduct the experiment, after RL training, we deploy all models on a set of 140 held-out simple games. Table 7.3 shows the results, where "Medium 100%" ("Hard 100%") denotes that the model is trained on medium (hard) games for the whole RL phase. The generalizability of GATA, which is trained purely with medium and hard games, is significantly low and cannot perform well on simple games. In contrast, our model performs very well and achieves over 80% of the scores. The world-perceiving modules, which are pre-trained with simple games, help to train a decision module that adapts well on unseen games. It is not surprising that the variant "IL w/o FT" also performs well on simple games, since they are only pre-trained with simple games. However, as indicated by the performance of "IL", after fine-tuning on medium/hard games (recalling Sec. 7.6.1), the action scorer "forgets" the experience/skills dealing with simple games and the model fails to generalize on unseen simple games. In summary, the best performance achieved by QWA demonstrates that our model can generalize well on games with different complexities.

7.6.3 Ablation Study

We study the contribution of the subtask time-awareness by comparing our full model with the variant without this technique. Fig. 7.5 shows the result. Although the models perform similarly in the medium games, the full model shows better performance in the hard games, where there may exist more difficult subtasks (we regard a subtask more difficult if it requires more actions to be completed). Assigning each subtask a time limit prevents the agent from pursuing a too difficult subtask, and improves subtask diversity by encouraging the agent to try different subtasks. Besides,

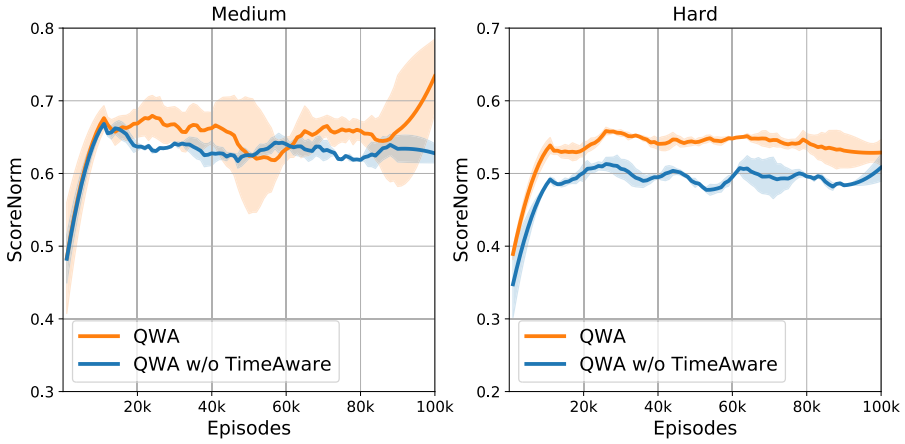


FIGURE 7.5: The performance of our model and the variant without time-awareness.

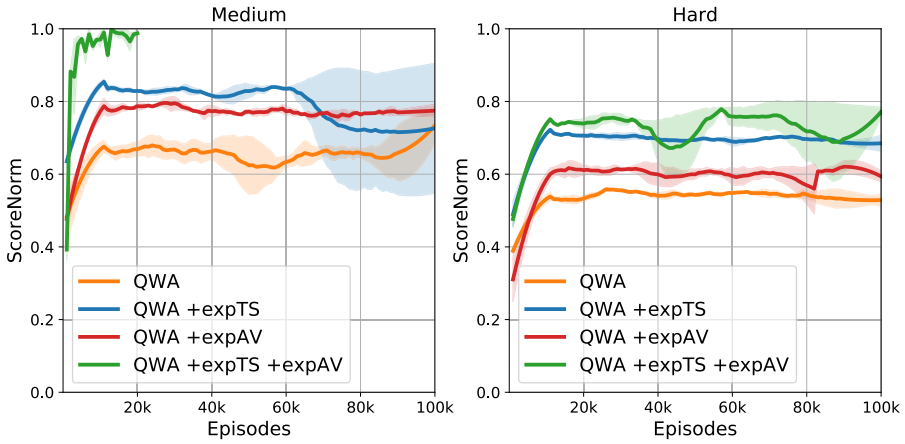


FIGURE 7.6: The performance of our model and the variants with expert modules.

it prevents the agent from being stuck in a wrong subtask, making the agent more robust to the compound error.

We then investigate the performance upper bound of our method by comparing our model to variants with oracle world-perceiving modules. Fig. 7.6 shows the results, where "+expTS" ("+expAV") denotes that the model uses an expert task selector (action validator). There's still space to improve the pre-trained modules. The variant "QWA +expTS +expAV" solves all the medium games and achieves nearly 80% of the scores in hard games, showing the potential of introducing world-perceiving modules in facilitating RL. We also find that assigning either the expert task selector or the expert action validator helps to improve the performance. In light of these findings, we will consider more powerful pre-training methods as a future direction.

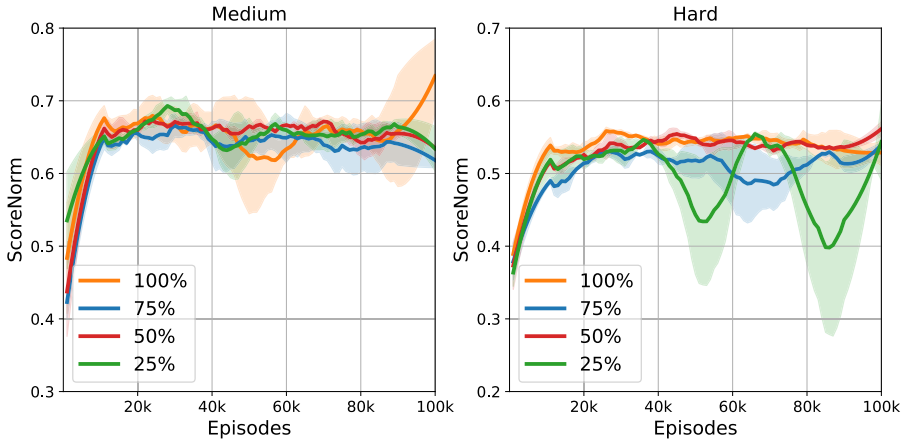


FIGURE 7.7: The performance of our model with varying amounts of pre-training data.

7.6.4 Pre-training on the Partial Dataset

Although we only collect labeled data from the simple games, it is still burdensome for human players to go through the games and answer the questions. We are thus curious about how the performance of QWA (or world-perceiving modules) varies with the amount of pre-training data being reduced. Fig. 7.7 shows the results, where the pre-training dataset has been reduced to 75%, 50% and 25%, respectively. Our model still works well when the pre-training data is reduced to 75% and 50%. When we only use 25% of the pre-training data, the model exhibits instability during the learning of hard games, while its final performance is still comparable. To summarize, our model is robust to limited pre-training data and largely alleviates the burden of human annotations.

7.7 Conclusion

In this chapter, we addressed the challenges of low sample efficiency and large action space for language-conditional reinforcement learning. We introduced the world-perceiving modules, which are capable of automatic task decomposition and action pruning through answering questions about the environment. We proposed a two-phase training framework, which decouples the language learning from the reinforcement learning. We empirically demonstrated that the proposed method not only achieved improved performance with high sample efficiency, but also exhibited robustness against compound error and limited pre-training data.

With respect to the limitations & future directions, our QWA is designed with some simplifications, which do not hold in real world applications. For example, we simplify the pre-training tasks as binary classification problems, while the recent progress of multi-label learning [233], could be leveraged to yield better performance. Another limitation is that our world-perceiving modules are very simple, while the large pre-trained language models, and more advanced pre-training methods, could be incorporated with our model. Besides, this work only consider two types of world-perceiving modules, and we would like to investigate such modules with other functionalities in the future works. Last but not least, we are willing to explore the potential of QWA in other tasks, such as commonsense reasoning [147] and systematic generalization [169].

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The language-conditional RL, which aims to ground natural language into reinforcement learning, has attracted increasing attention in recent years. While studying this cross-domain topic brings benefits to both natural language processing and reinforcement learning, new challenges arise and hinder the further development of this domain. This thesis studies the language-condition RL, and would like to provide solutions to the challenges. We center on the setting where the observation space as well as the action space is language-based, while the setting of language-based instruction and reward function will also be partly involved. We summarize our main contributions below:

In Chapter 4, we explored the potential of the transformer architecture for building state representations in solving the language-conditional reinforcement learning tasks. We designed an adaptable transformer-based representation generator equipped with three modifications. We first conducted layer normalization reordering within the blocks. Then we shared weights among the blocks. Finally, we applied the gate aggregation between the blocks. We empirically validated our method on both synthetic and man-made text-based games with different settings. The proposed method showed higher sample efficiency in solving single synthetic games, better generalizability in solving unseen synthetic games, and better performance in solving complex man-made games.

In Chapter 5, we studied the reasoning process in the language-conditional reinforcement learning. We designed the SHA-KG agent, which is empowered with the reasoning ability over multi-modal

inputs through two techniques: the knowledge graph division, and the stacked hierarchical attention mechanism. Besides demonstrating the effectiveness of our agent in solving a range of man-made text-based games, we also interpreted how the reasoning process is conducted by SHA-KG in deriving the actions.

In Chapter 6, we investigated generalization for language-conditional RL from the perspective of solving text-based games. We introduced a two-level hierarchical framework, H-KGA, to tackle the generalization problem. In the high level, we considered a meta-policy for task decomposition and subtask selection. Then, in the low level, we considered a sub-policy for subtask-conditioned action selection. Besides, we proposed two techniques, the scheduled task sampling strategy, and the level-aware replay buffers, to facilitate training under the multi-task learning setting. We empirically validated the effectiveness of H-KGA upon 8 game sets with different levels and generalization types.

In Chapter 7, we addressed the challenges of low sample efficiency and large action space for language-conditional reinforcement learning. We introduced the world-perceiving modules, which are capable of automatic task decomposition and action pruning through answering questions about the environment. We proposed a two-phase training framework, which decouples the language learning from the reinforcement learning. We empirically demonstrated that the proposed method not only achieved improved performance with high sample efficiency, but also exhibited robustness against compound error and limited pre-training data.

8.2 Future Work

We list the future directions as follows:

- State representation learning: The models proposed in this research either considered the single game setting (Chapter 4 and Chapter 5), or assumed that all games are with similar vocabulary (Chapter 4, Chapter 6 and Chapter 7). It's worth investigating how to leverage the pre-trained language models [37, 164] / knowledge graph models [100, 185] to obtain better state representations, thus achieving better cross-domain generalizability and commonsense reasoning ability.

- Pre-training techniques: In Chapter 7, we introduced the world-perceiving modules, which are obtained through supervised pre-training. We noticed that there's still a large gap between the pre-trained modules and the oracle modules. As an ongoing work, we would like to investigate how the performance could be further improved by integrating advanced pre-training techniques, such as the contrastive learning objective [222], and the KG-based data augmentation [237].

- Real world applications: We still used the simulated game environment as the test-beds. In the future work, we would like to apply our methods in more practical scenarios. For example, our "game playing" agents could be served as the virtual assistants in the task-oriented dialogue system, and help users to achieve specific tasks by providing textual guidance [47].

- Off-line RL: For real world applications, due to the cost and safety concerns, it's impractical to train an RL agent by letting it interact with the environment. As a future direction, we would like to investigate how to design language-conditional RL agent in the offline setting, i.e. training the agent with a fixed offline dataset (e.g., user logs), instead of an interactive environment [123].

Appendix A

Appendix for Chapter 6

We provide more game examples to demonstrate the difference in layouts and difficulty levels. Fig. A.1 visualizes the initial observation o_0^{KG} for four games, where "S1 Game1" and "S1 Game2" belong to level "S1", "S2 Game1" and "S2 Game2" belong to level "S2". Fig. A.2 visualizes the initial observation of two games belonging to level "S3". Fig. A.3 visualizes the initial observation of one game belonging to level "S4". Games within the same level have the same complexity, but are different in their layouts. For example, "S2 Game1" and "S2 Game2" have the same number of rooms, ingredients and requirements, but are different in the type of the ingredients and the requirements. Similarly, "S3 Game1" and "S3 Game2" have the same number of rooms, but are with different room connectivity. Games within different levels have different complexities, and their layouts are naturally different (e.g., "S1 Game1" v.s., "S2 Game1" v.s., "S3 Game1" v.s., "S4 Game1"). The TextWorld also provides other game themes, such as the default House theme, the Coin Collector theme and the Treasure Hunter theme. In these themes, the agent needs to go through the rooms to find either the coin, or an object specified at the beginning of an episode. We believe that the Cooking theme we use in this work is able to cover these themes, as it requires the agent to navigate through different numbers of rooms, collect different numbers of ingredients, and prepare them in different ways.

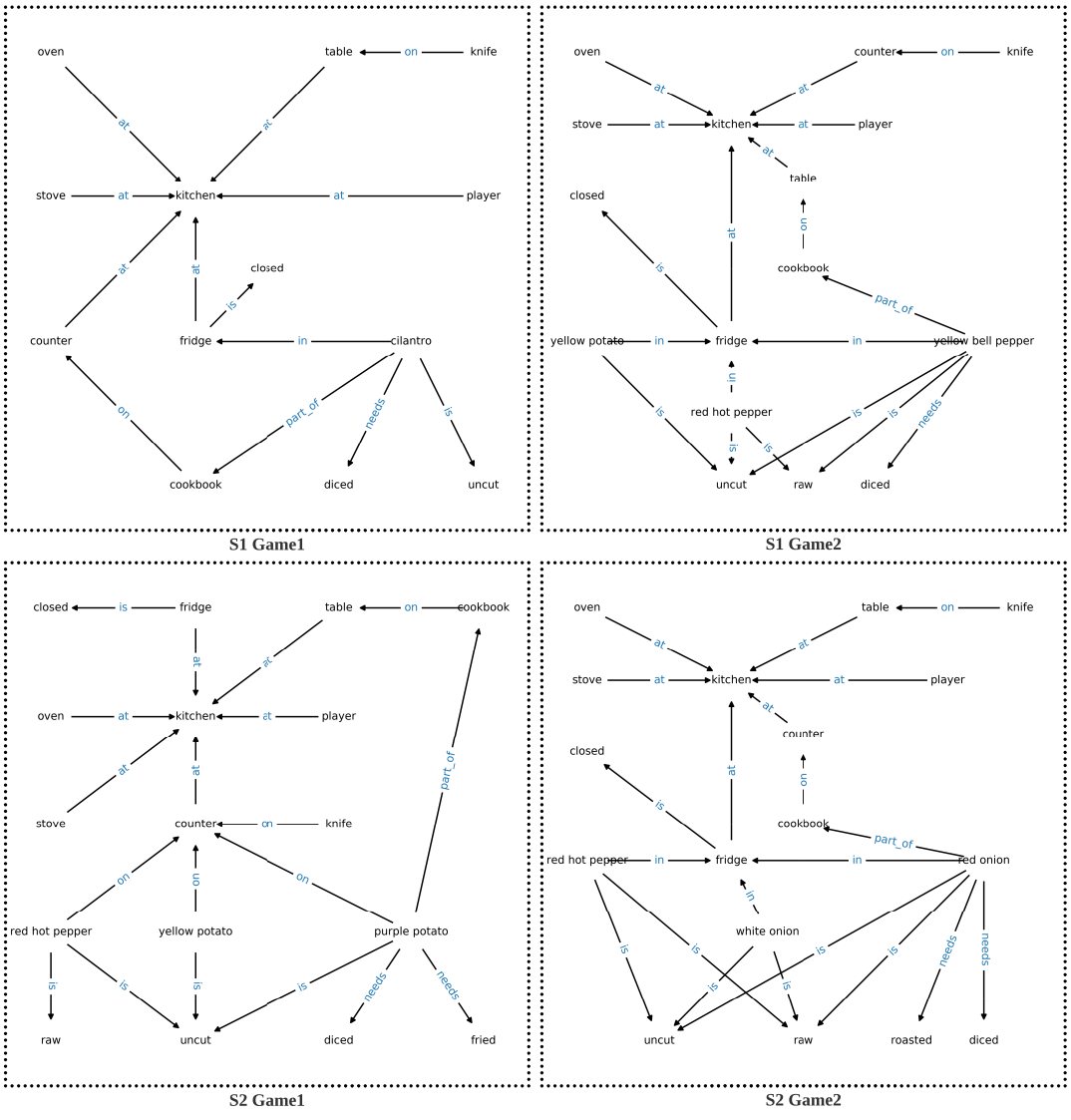
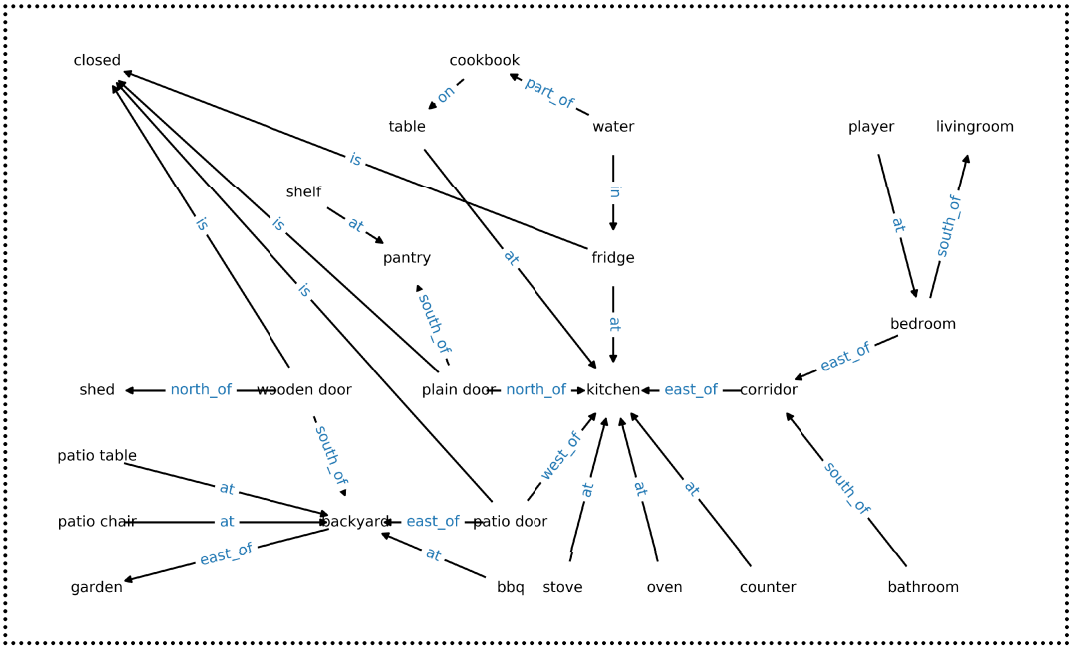
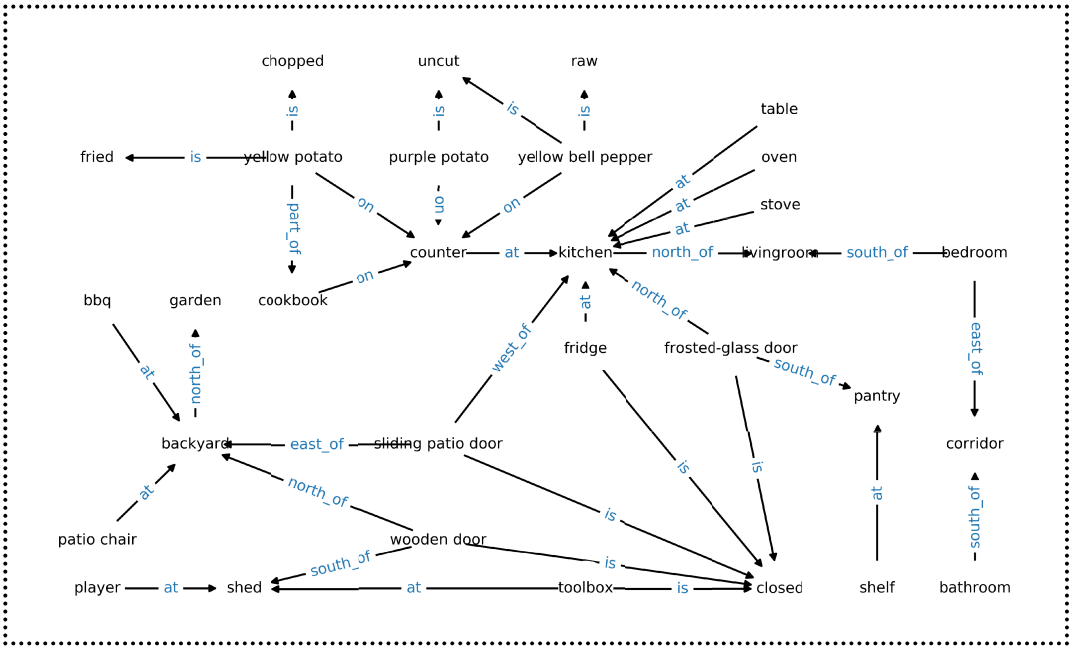


FIGURE A.1: The initial observation of four games, where "S1 Game1" and "S1 Game2" belong to level "S1", "S2 Game1" and "S2 Game2" belong to level "S2".



S3 Game1



S3 Game2

FIGURE A.2: The initial observation of two games belonging to level "S3".

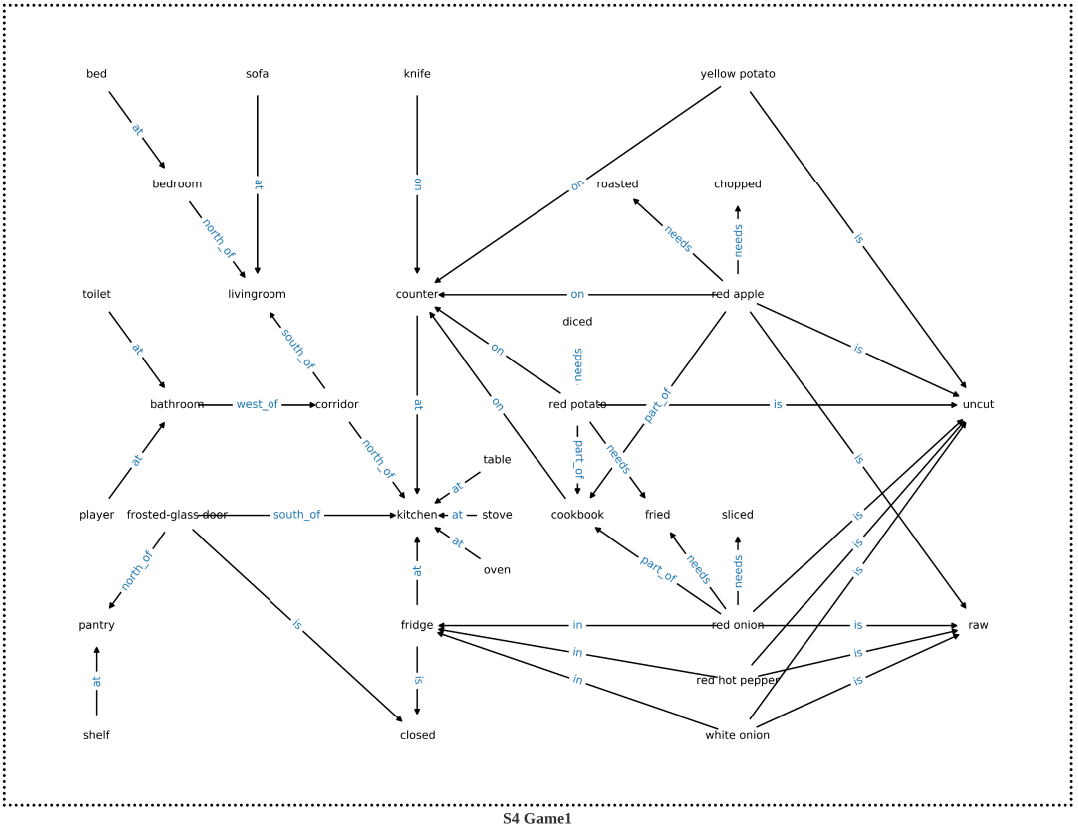


FIGURE A.3: The initial observation of one game belonging to level "S4".

Appendix B

Appendix for Chapter 7

The appendix for Chapter 7 is organized as follows: Sec. B.1 details the environment. Sec. B.2 illustrates the process for constructing the pre-training datasets. Sec. B.3 demonstrates the baselines' architecture and training details. Sec. B.4 provides more experimental results.

B.1 Environment

In the cooking game [53], the player is located in a house, which contains multiple rooms and interactive objects (food, tools, etc.). Her / his task is to follow the recipe to prepare the meal. Each game instance has a unique recipe, including different numbers of ingredients (food objects that are necessary for preparing the meal) and their corresponding preparation requirements (e.g., "slice", "fry"). Besides the textual observation, the KG-based observation can also be directly obtained from the environment. The game sets used in our work contains a task set $\mathcal{T}$ of 268 subtasks, and an action set $\mathcal{A}$ of 1304 actions. Following GATA's experiment setting [2], we simplify the game environment by making the action set changeable over time, which can be provided by the TextWorld platform. Note that although the action space is reduced, it still remains challenging as the agent may encounter unseen action candidates [42, 43]. We then use a similar way to obtain a changeable task set, which is a combination of the verb set {chop, dice, slice, fry, make, get, grill, roast} and the ingredient set. Table B.1 and Table B.2 show the KG-based observations o_t, corresponding subtask candidates $\mathcal{T}$ and action candidates $\mathcal{A}$. Table B.3 and Table B.4 show more examples of

TABLE B.1: The observations o_t, subtask candidates $\mathcal{T}$ and action candidates $\mathcal{A}$ of a simple game and a medium game. The underlined subtask candidates denote the available subtask set $\mathcal{T}_t$.

Game	KG-based observation	Subtask candidates	Action candidates
Simple	["cheese block", "cookbook", "part_of"], ["cheese block", "fried", "needs"], ["cheese block", "player", "in"], ["cheese block", "raw", "is"], ["cheese block", "sliced", "needs"], ["cheese block", "uncut", "is"], ["cookbook", "counter", "on"], ["counter", "kitchen", "at"], ["fridge", "kitchen", "at"], ["fridge", "open", "is"], ["knife", "counter", "on"], ["oven", "kitchen", "at"], ["player", "kitchen", "at"], ["stove", "kitchen", "at"], ["table", "kitchen", "at"]	"fry cheese block", "get knife", "chop cheese block", "dice cheese block", "get cheese block", "grill cheese block", "make meal", "roast cheese block", "slice cheese block"	"close fridge", "cook cheese block with oven", "cook cheese block with stove", "drop cheese block", "eat cheese block", "insert cheese block into fridge", "prepare meal", "put cheese block on counter", "put cheese block on stove", "put cheese block on table", "take cookbook from counter", "take knife from counter"
Medium	["bathroom", "corridor", "south_of"], ["bed", "bedroom", "at"], ["bedroom", "livingroom", "north_of"], ["cheese block", "cookbook", "part_of"], ["cheese block", "diced", "is"], ["cheese block", "diced", "needs"], ["cheese block", "fridge", "in"], ["cheese block", "fried", "is"], ["cheese block", "fried", "needs"], ["carrot", "fridge", "in"], ["carrot", "raw", "is"], ["carrot", "uncut", "is"], ["cookbook", "counter", "on"], ["corridor", "bathroom", "north_of"], ["corridor", "kitchen", "east_of"], ["corridor", "livingroom", "south_of"], ["counter", "kitchen", "at"], ["flour", "cookbook", "part_of"], ["flour", "shelf", "on"], ["fridge", "closed", "is"], ["fridge", "kitchen", "at"], ["frosted-glass door", "closed", "is"], ["frosted-glass door", "kitchen", "west_of"], ["frosted-glass door", "pantry", "east_of"], ["kitchen", "corridor", "west_of"], ["knife", "counter", "on"], ["livingroom", "bedroom", "south_of"], ["livingroom", "corridor", "north_of"], ["oven", "kitchen", "at"], ["parsley", "fridge", "in"], ["parsley", "uncut", "is"], ["player", "kitchen", "at"], ["pork chop", "chopped", "is"], ["pork chop", "chopped", "needs"], ["pork chop", "cookbook", "part_of"], ["pork chop", "fridge", "in"], ["pork chop", "fried", "is"], ["pork chop", "fried", "needs"], ["purple potato", "counter", "on"], ["purple potato", "uncut", "is"], ["red apple", "counter", "on"], ["red apple", "raw", "is"], ["red apple", "uncut", "is"], ["red onion", "fridge", "in"], ["red onion", "raw", "is"], ["red onion", "uncut", "is"], ["red potato", "counter", "on"], ["red potato", "uncut", "is"], ["shelf", "pantry", "at"], ["sofa", "livingroom", "at"], ["stove", "kitchen", "at"], ["table", "kitchen", "at"], ["toilet", "bathroom", "at"], ["white onion", "fridge", "in"], ["white onion", "raw", "is"], ["white onion", "uncut", "is"]	"get cheese block", "get flour", "get pork chop", "chop cheese block", "chop flour", "chop pork chop", "dice cheese block", "dice flour", "dice pork chop", "fry cheese block", "fry flour", "fry pork chop", "get knife", "grill cheese block", "grill flour", "grill pork chop", "make meal", "roast cheese block", "roast flour", "roast pork chop", "slice cheese block", "slice flour", "slice pork chop"	"go east", "open fridge", "open frosted-glass door", "take cookbook from counter", "take knife from counter", "take purple potato from counter", "take red apple from counter", "take red potato from counter"

subtasks and actions, respectively. The underlined subtask candidates denote the available subtask set $\mathcal{T}_t$. The underlined action candidates in Table B.4 denote the refined action set $\mathcal{A}_t$ after selecting the subtask "roast carrot". We still denote the subtask candidate set (action candidate set) as $\mathcal{T}$ ($\mathcal{A}$) to distinguish it from the available subtask set $\mathcal{T}_t$ (refined action set $\mathcal{A}_t$).

TABLE B.2: The observations o_t, subtask candidates $\mathcal{T}$ and action candidates $\mathcal{A}$ of a hard game. The underlined subtask candidates denote the available subtask set $\mathcal{T}_t$. The underlined action candidates denote the refined action set $\mathcal{A}_t$ after selecting the subtask "roast carrot".

Game	KG-based observation	Subtask candidates	Action candidates
Hard	["backyard", "garden", "west_of"], ["barn door", "backyard", "west_of"], ["barn door", "closed", "is"], ["barn door", "shed", "east_of"], ["bathroom", "corridor", "east_of"], ["bbq", "backyard", "at"], ["bed", "bedroom", "at"], ["bedroom", "corridor", "north_of"], ["bedroom", "livingroom", "south_of"], ["carrot", "cookbook", "part_of"], ["carrot", "player", "in"], ["carrot", "raw", "is"], ["carrot", "roasted", "needs"], ["carrot", "sliced", "needs"],["carrot", "uncut", "is"], ["commercial glass door", "closed", "is"], ["commercial glass door", "street", "east_of"], ["commercial glass door", "supermarket", "west_of"], ["cookbook", "table", "on"], ["corridor", "bathroom", "west_of"], ["corridor", "bedroom", "south_of"], ["counter", "kitchen", "at"], ["driveway", "street", "north_of"], ["fridge", "closed", "is"], ["fridge", "kitchen", "at"], ["front door", "closed", "is"], ["front door", "driveway", "west_of"], ["front door", "livingroom", "east_of"], ["frosted-glass door", "closed", "is"], ["frosted-glass door", "kitchen", "south_of"], ["frosted-glass door", "pantry", "north_of"], ["garden", "backyard", "east_of"], ["kitchen", "livingroom", "west_of"], ["knife", "counter", "on"], ["livingroom", "bedroom", "north_of"], ["livingroom", "kitchen", "east_of"], ["oven", "kitchen", "at"], ["patio chair", "backyard", "at"], ["patio door", "backyard", "north_of"], ["patio door", "corridor", "south_of"], ["patio door", "open", "is"], ["patio table", "backyard", "at"], ["player", "backyard", "at"], ["red apple", "counter", "on"], ["red apple", "raw", "is"], ["red apple", "uncut", "is"], ["red hot capsicum", "cookbook", "part_of"], ["red hot capsicum", "player", "in"], ["red hot capsicum", "raw", "is"], ["red hot capsicum", "roasted", "needs"], ["red hot capsicum", "sliced", "needs"], ["red hot capsicum", "uncut", "is"], ["red onion", "garden", "at"], ["red onion", "raw", "is"], ["red onion", "uncut", "is"], ["shelf", "pantry", "at"], ["showcase", "supermarket", "at"], ["sofa", "livingroom", "at"], ["stove", "kitchen", "at"], ["street", "driveway", "south_of"], ["table", "kitchen", "at"], ["toilet", "bathroom", "at"], ["toolbox", "closed", "is"], ["toolbox", "shed", "at"], ["white onion", "chopped", "needs"], ["white onion", "cookbook", "part_of"], ["white onion", "grilled", "needs"], ["white onion", "player", "in"], ["white onion", "raw", "is"], ["white onion", "uncut", "is"], ["workbench", "shed", "at"], ["yellow bell capsicum", "garden", "at"], ["yellow bell capsicum", "raw", "is"], ["yellow bell capsicum", "uncut", "is"]	"roast carrot", "roast red hot capsicum", "grill white onion", "get knife", "chop carrot", "chop red hot capsicum", "chop white onion", "dice carrot", "dice red hot capsicum", "dice white onion", "fry carrot", "fry red hot capsicum", "fry white onion", "get carrot", "get red hot capsicum", "get white onion", "grill carrot", "grill red hot capsicum", "make meal", "roast white onion", "slice carrot", "slice red hot capsicum", "slice white onion"	"go east", "go north", "open barn door", "open patio door", "close patio door", "cook carrot with bbq", "cook red hot capsicum with bbq", "cook white onion with bbq", "drop carrot", "drop red hot capsicum", "drop white onion", "eat carrot", "eat red hot capsicum", "eat white onion", "put carrot on patio chair", "put carrot on patio table", "put red hot capsicum on patio chair", "put red hot capsicum on patio table", "put white onion on patio chair", "put white onion on patio table"

TABLE B.3: Examples of subtasks.

Subtask candidates		
chop banana	chop black capsicum	chop cheese block
chop olive oil	chop orange bell capsicum	chop parsley
chop vegetable oil	chop water	chop white onion
dice cilantro	dice egg	dice flour
dice red bell capsicum	dice red hot capsicum	dice red onion
dice yellow potato	fry banana	fry black capsicum
fry milk	fry olive oil	fry orange bell capsicum
fry tomato	fry vegetable oil	fry water
get chicken wing	get cilantro	get egg
get purple potato	get red apple	get red bell capsicum
get yellow bell capsicum	get yellow onion	get yellow potato
grill green hot capsicum	grill lettuce	grill milk
grill salt	grill sugar	grill tomato
roast carrot	roast chicken breast	roast chicken leg
roast peanut oil	roast pork chop	roast purple potato
roast white tuna	roast yellow apple	roast yellow bell capsicum
slice green apple	slice green bell capsicum	slice green hot capsicum
slice red potato	slice red tuna	slice salt

TABLE B.4: Examples of actions.

Action candidates		
chop banana with knife	chop cheese block with knife	chop carrot with knife
cook cheese block with oven	cook cheese block with stove	cook carrot with bbq
cook orange bell capsicum with oven	cook orange bell capsicum with stove	cook parsley with bbq
cook water with stove	cook white onion with bbq	cook white onion with oven
drink water	drop banana	drop black capsicum
eat carrot	eat chicken breast	eat chicken leg
insert cheese block into toolbox	insert carrot into fridge	insert carrot into toolbox
insert red onion into fridge	insert red onion into toolbox	insert red potato into fridge
put banana on shelf	put banana on showcase	put banana on sofa
put chicken breast on showcase	put chicken breast on sofa	put chicken breast on stove
put egg on patio table	put egg on shelf	put egg on showcase
put green hot capsicum on shelf	put green hot capsicum on showcase	put green hot capsicum on sofa
put olive oil on patio chair	put olive oil on patio table	put olive oil on shelf
put pork chop on sofa	put pork chop on stove	put pork chop on table
put red hot capsicum on table	put red hot capsicum on toilet	put red hot capsicum on workbench
put salt on workbench	put sugar on bed	put sugar on counter
put white onion on shelf	put white onion on showcase	put white onion on sofa
put yellow onion on sofa	put yellow onion on stove	put yellow onion on table
take banana from patio chair	take banana from patio table	take banana from shelf
take carrot from showcase	take carrot from sofa	take carrot from stove
take chicken wing from toolbox	take chicken wing from workbench	take cilantro
take green apple from bed	take green apple from counter	take green apple from fridge
take lettuce from sofa	take lettuce from stove	take lettuce from table
take orange bell capsicum from workbench	take parsley	take parsley from bed
take purple potato from showcase	take purple potato from sofa	take purple potato from stove
take red hot capsicum from toolbox	take red hot capsicum from workbench	take red onion
take salt from counter	take salt from fridge	take salt from patio chair
take water from counter	take water from fridge	take water from patio chair
take yellow apple from sofa	take yellow apple from stove	take yellow apple from table

B.2 Pre-training Datasets

We build separate datasets for each pre-training task (task decomposition, action pruning, and imitation learning). We first let the player to go through each simple game, then construct the datasets upon the interaction data. For each time step, the game environment provides the player with the action set $\mathcal{A}$ and the KG-based observation o_t, which is represented as a set of triplets. We use a simple method to build the subtask set $\mathcal{T}$ from o_t: As shown in Fig. B.1, we first obtain the ingredients by extracting the nodes having the relation "part_of" with the node "cookbook". Then we build $\mathcal{T}$ as the Cartesian product of the ingredients and the verbs {chop, dice, slice, fry, get, grill, roast} plus two special subtasks "get knife" and "make meal". The player is required to select a subtask $T_t \in \mathcal{T}$, and select an action $a_t \in \mathcal{A}$. After executing a_t, the environment will transit to next state s_{t+1}, and the player will receive o_{t+1} and r_{t+1} to form a transition $\{o_t, \mathcal{T}, T_t, \mathcal{A}, a_t, o_{t+1}, r_{t+1}\}$, where $\{o_t, \mathcal{T}, T_t, \mathcal{A}, a_t\}$ will be used for imitation learning. Fig. B.1 shows the construction process of the pre-training dataset for task decomposition. Each subtask candidate $T \in \mathcal{T}$ will formulate a question "Is T available?", whose answer is 1 (yes) if T is an available subtask for o_t, otherwise 0 (no). Fig. B.2 shows the construction process of the pre-training dataset for action pruning. The action selector is made invariant of o_t, that we consider every subtask candidate $T \in \mathcal{T}$ during pre-training, regardless of whether T is a currently-available subtask. Each action candidate $a \in \mathcal{A}$ will be paired with T to formulate a question "Is a relevant to T", whose answer is 1 if a is relevant to T, otherwise 0.

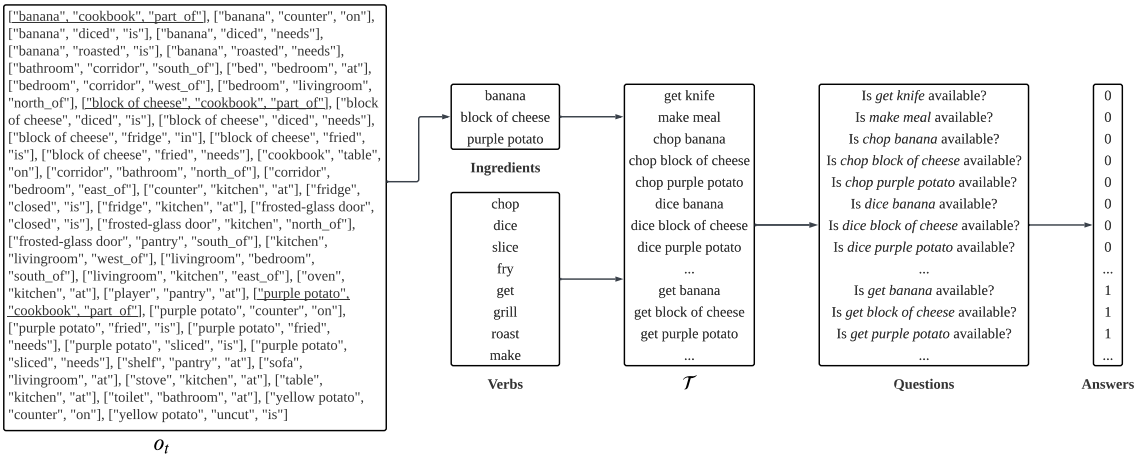


FIGURE B.1: The construction process of the subtask set $\mathcal{T}$, and the pre-training dataset for task decomposition.

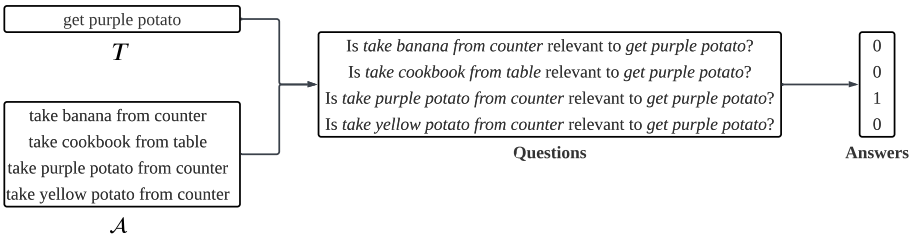


FIGURE B.2: The construction process of the pre-training dataset for action pruning.

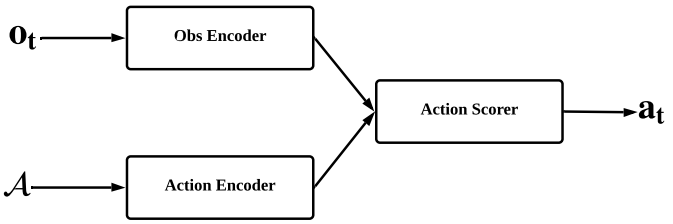


FIGURE B.3: The architecture of GATA baseline.

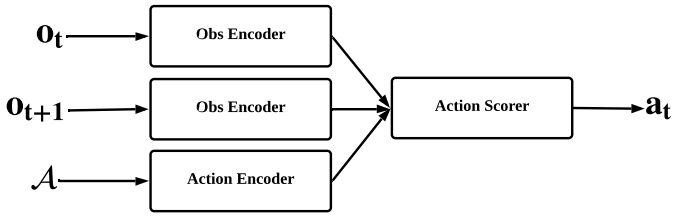


FIGURE B.4: The architecture of GATA for action prediction.

B.3 Baseline details

B.3.1 GATA

Fig. B.3 shows our building backbone GATA, which consists of an observation encoder, an action encoder and an action scorer. The observation encoder is a graph encoder for encoding the KG-based observation o_t. The action encoder is a text encoder for encoding the action set $\mathcal{A}$ as a stack of action candidate representations. The observation representation will be paired with each action candidate, and then fed into the action scorer, which consists of linear layers.

We train the GATA through reinforcement learning, the experiment setting is same with Sec. 7.5.3. Instead of initializing the word embedding, node embedding and edge embedding with fastText word vectors [139], we found that the action prediction task (AP), which is also included in GATA's work [2], could provide better initialization. In light of this, we could like to conduct such task, and apply the AP initialization to all encoders (observation encoder, task encoder, action encoder). Fig. B.4 shows the action predicting process, where the task is to predict the action $a_t \in \mathcal{A}$ given o_t and o_{t+1}. The transition data for AP task is collected from the FTWP game set and is provided by GATA's released code.

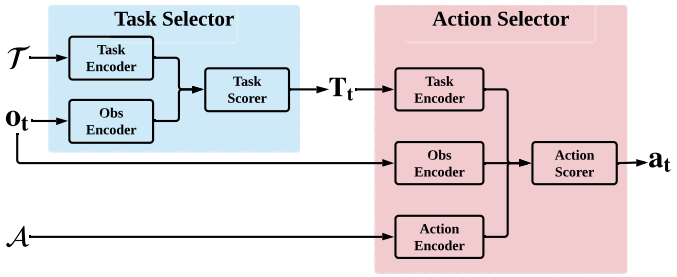


FIGURE B.5: The architecture of IL baseline.

B.3.2 IL

Fig. B.5 shows the IL baseline. We follow [49] to conduct a two-phase training process: imitation pre-training and reinforcement fine-tuning. In the imitation pre-training phase, we use the transition data to train both the task selector ($f(o_t, \mathcal{T}) \rightarrow T_t$) and the action selector ($f(o_t, T_t, \mathcal{A}) \rightarrow a_t$) through supervised learning. The modules are optimized via cross entropy loss and Adam optimizer with learning rate 0.001. We train the modules with batch size 128 for up to 50 epochs. Then in the reinforcement fine-tuning phase, we freeze the task selector and fine-tune the action selector through reinforcement learning, where the experiment setting is same with QWA and GATA.

B.4 More experimental results

In the pre-training phase, we conduct rough hyper-parameter tuning by varying batch sizes. Fig. B.6 and Fig. B.7 show the pre-training performance of QWA's task selector and action validator, respectively. Fig. B.8 shows the pre-training performance of IL baseline.

Fig. B.9 compares our GATA and the original GATA without the action prediction initialization. Fig. B.10, Fig. B.11, Fig. B.12 and Fig. B.13 show the full results of Fig. 7.4, Fig. 7.5, Fig. 7.6 and Fig. 7.7, respectively.

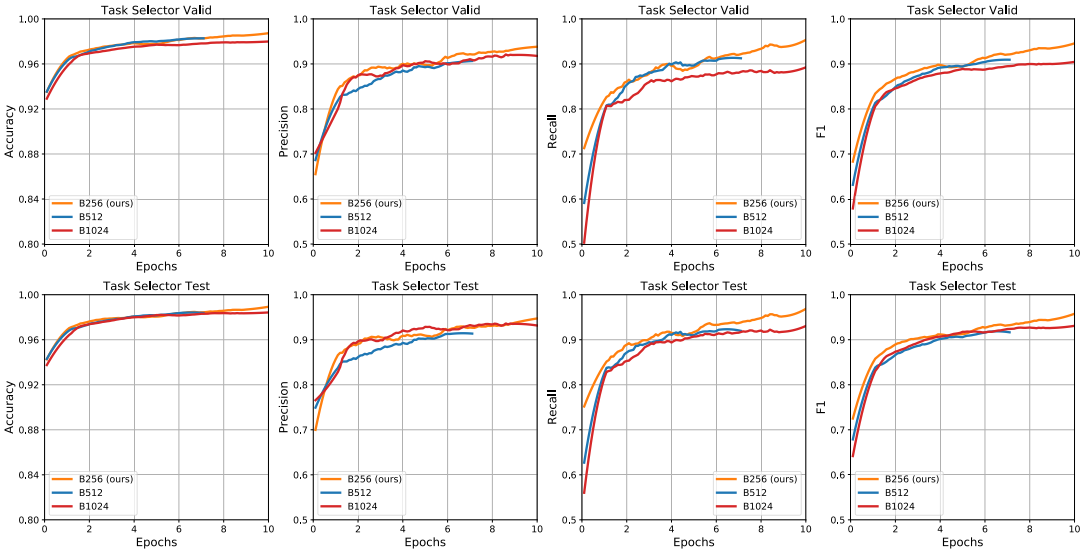


FIGURE B.6: The pre-training performance of QWA's task selector. The results are averaged by 3 random seeds, we omit the standard deviation as the performance is relatively stable.

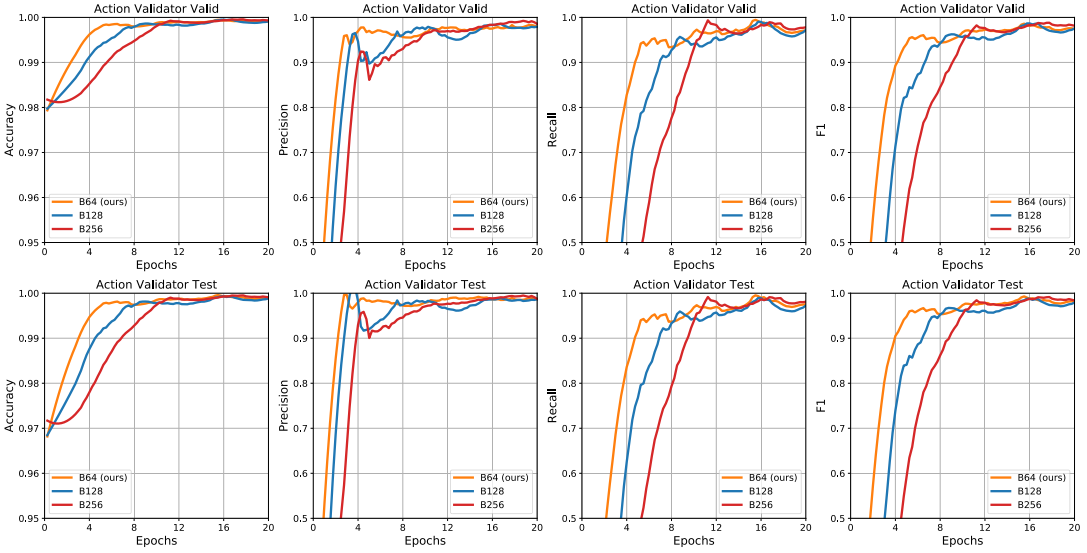


FIGURE B.7: The pre-training performance of QWA's action validator.

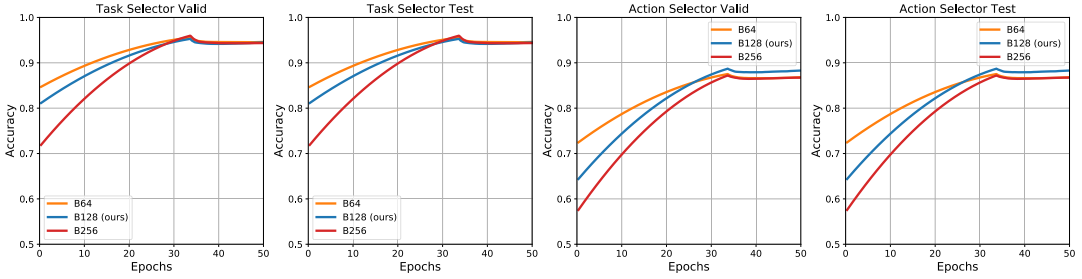


FIGURE B.8: The pre-training performance of IL's task selector and action selector.

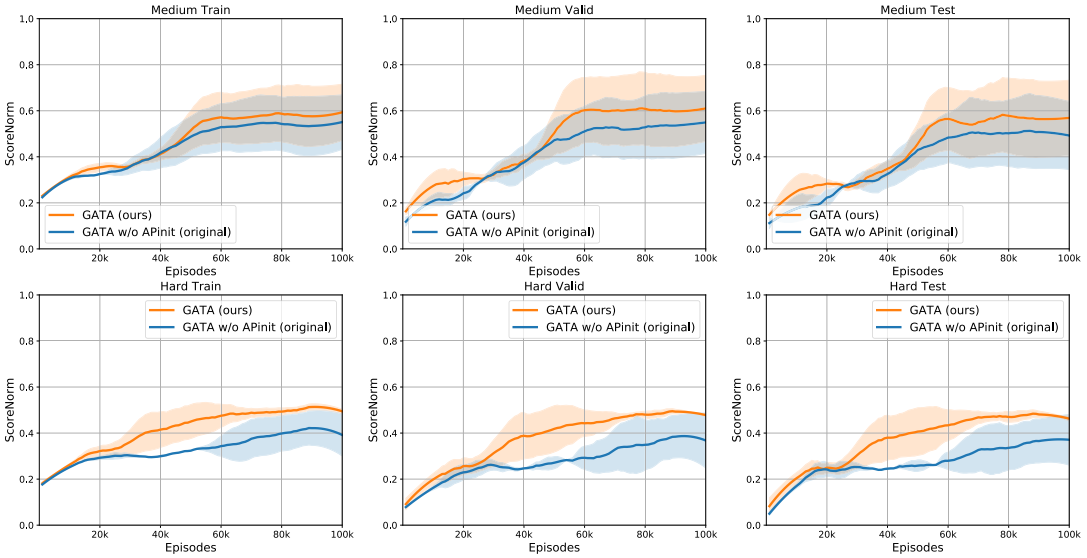


FIGURE B.9: The RL performance of our GATA baseline and the original GATA without AP initialization.

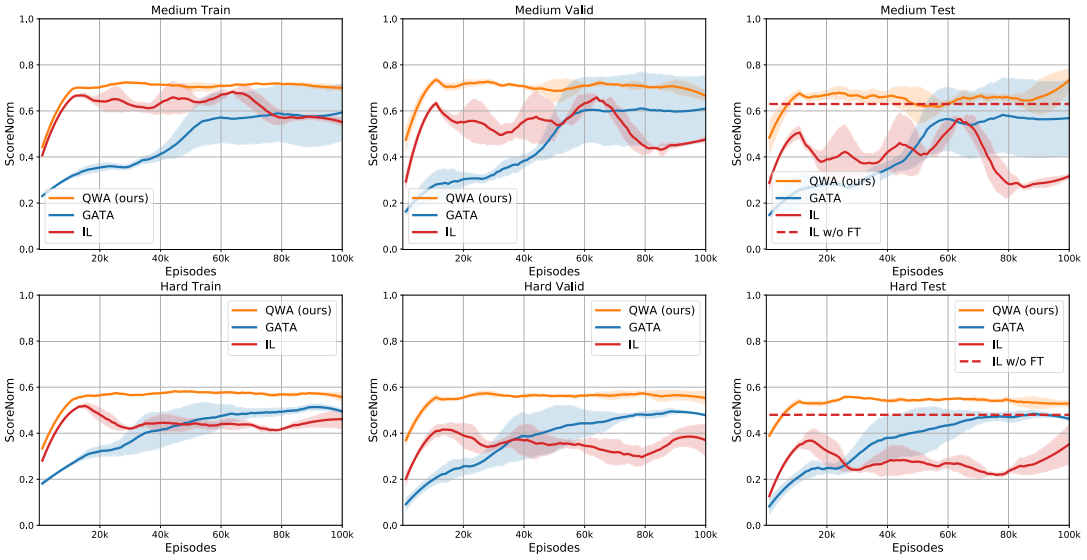


FIGURE B.10: The RL performance w.r.t. the training episodes (the full result of Fig. 7.4).

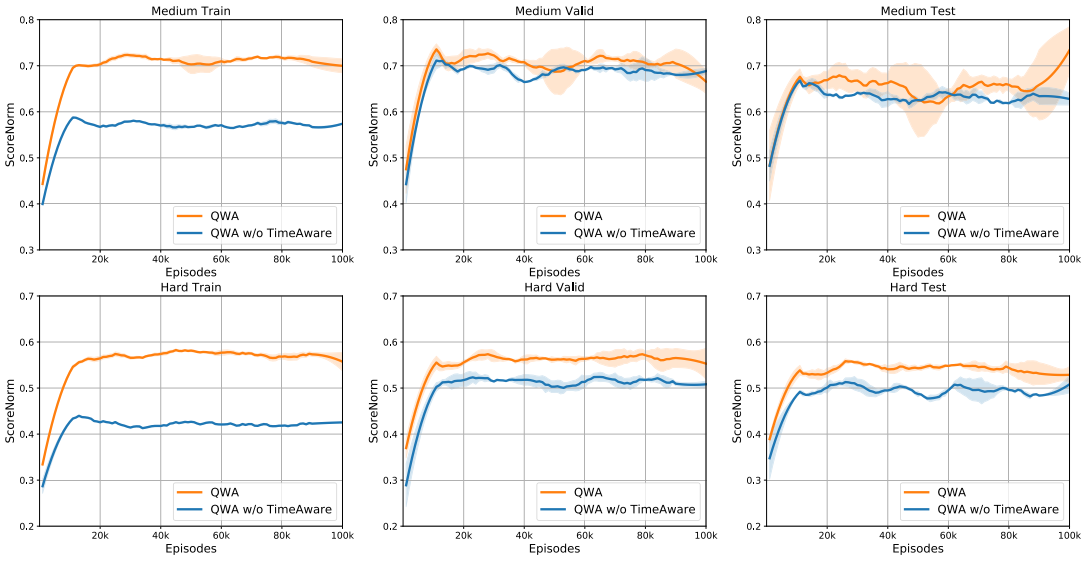


FIGURE B.11: The RL performance of our agent and the variant without time-awareness (the full result of Fig. 7.5).

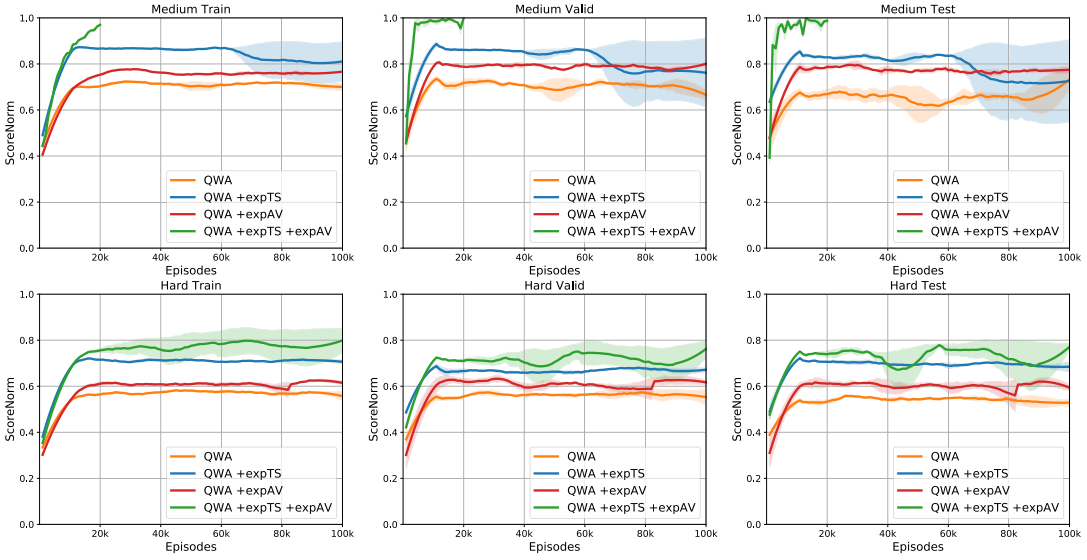


FIGURE B.12: The performance of our agent and the variants with expert modules (the full result of Fig. 7.6).

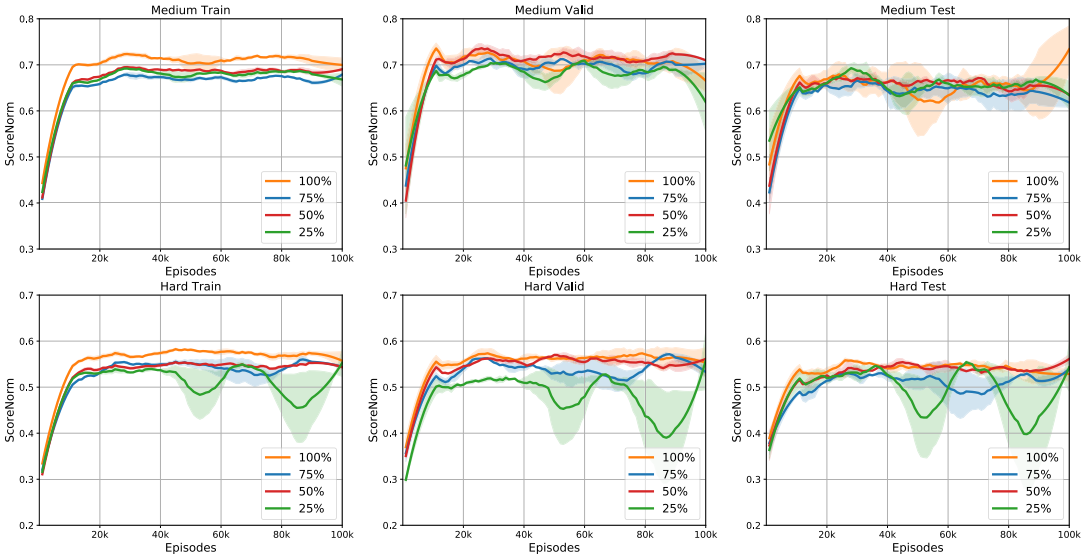


FIGURE B.13: The performance of our agent with varying amounts of pre-training data (the full result of Fig. 7.7).

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *International Conference on Machine Learning (ICML)*, pages 22–31. PMLR.

[2] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. 2020. Learning dynamic belief graphs to generalize on text-based games. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 3045–3057.

[3] Leonard Adolphs and Thomas Hofmann. 2020. Ledeepchef: Deep reinforcement learning agent for families of text-based games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 7342–7349.

[4] James Allen. 1988. *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc.

[5] Eyal Amir and Allen Chang. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402.

[6] Prithviraj Ammanabrolu, William Broniec, Alex Mueller, Jeremy Paul, and Mark O Riedl. 2019. Toward automated quest generation in text-adventure games. In *Proceedings of the 4th Workshop on Computational Creativity in Language Generation*, pages 1–12.

[7] Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations (ICLR)*.

[8] Prithviraj Ammanabrolu and Mark Riedl. 2019. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, volume 1, pages 3557–3565.

[9] Prithviraj Ammanabrolu and Mark Riedl. 2019. Transfer in deep reinforcement learning using knowledge graphs. In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, pages 1–10.

[10] Prithviraj Ammanabrolu, Ethan Tien, Matthew Hausknecht, and Mark O Riedl. 2020. How to avoid being eaten by a grue: Structured exploration strategies for textual worlds. *arXiv preprint arXiv:2006.07409*.

[11] Prithviraj Ammanabrolu, Jack Urbanek, Margaret Li, Arthur Szlam, Tim Rocktäschel, and Jason Weston. 2021. How to motivate your dragon: Teaching goal-driven agents to speak and act in fantasy worlds. In *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 807–833.

[12] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning (ICML)*, volume 70, pages 166–175.

[13] Jacob Andreas, Dan Klein, and Sergey Levine. 2018. Learning with latent language. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2166–2179.

[14] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48. IEEE.

[15] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5048–5058.

[16] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 344–354.

[17] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2425–2433.

[18] Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

[19] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson LS Wong, and Stefanie Tellex. 2017. Accurately and efficiently interpreting human-robot instructions of varying granularities. In *Proceedings of Robotics: Science and Systems (RSS)*.

[20] Timothy Atkinson, Hendrik Baier, Tara Copplestone, Sam Devlin, and Jerry Swan. 2019. The text-based adventure ai competition. *IEEE Transactions on Games*, 11(3):260–266.

[21] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. 2019. Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations (ICLR)*.

[22] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. 2020. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087.

[23] Lawrence W Barsalou. 2008. Grounded cognition. *Annual Review of Psychology*, 59:617–645.

[24] Madeleine Bates. 1995. Models of natural language understanding. *Proceedings of the National Academy of Sciences*, 92:9977–9982.

[25] Lisa Bauer, Yicheng Wang, and Mohit Bansal. 2018. Commonsense for generative multi-hop question answering tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4220–4230.

[26] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 1471–1479.

[27] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[28] Richard Bellman. 1957. A markovian decision process. *Journal of Mathematical Mechanics*, 6:679–684.

[29] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *International Conference on Machine Learning (ICML)*, pages 41–48.

[30] Marc Blank and Mike Berlyn. 1997. Zork: The undiscovered underground. Infocom.

[31] Marc Blank and David Lebling. 1980. Zork i: The great underground empire. Infocom.

[32] Léon Bottou. 2014. From machine learning to machine reasoning. *Machine learning*, 94(2):133–149.

[33] Craig Boutilier, Alon Cohen, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. 2018. Planning and learning with stochastic action sets. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4674–4682.

[34] SRK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 82–90.

[35] SRK Branavan, David Silver, and Regina Barzilay. 2012. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704.

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

[37] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901.

[38] Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Wojciech Gajewski, Andrea Gesmundo, Neil Houlsby, and Wei Wang. 2018. Ask the right questions: Active question reformulation with reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[39] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*.

[40] Andres Campero, Roberta Raileanu, Heinrich Kuttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. 2020. Learning with amigo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations (ICLR)*.

[41] Inigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. 2017. A benchmarking environment for reinforcement learning based task oriented dialogue management. *arXiv preprint arXiv:1711.11023*.

[42] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. 2019. Learning action representations for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 941–950. PMLR.

[43] Yash Chandak, Georgios Theocharous, Chris Nota, and Philip Thomas. 2020. Lifelong learning with a changing action set. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 3373–3380.

[44] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. 2018. Gated-attention architectures for task-oriented language grounding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, page 2819–2826.

[45] Subhajit Chaudhury, Daiki Kimura, Kartik Talamadupula, Michiaki Tatsubori, Asim Munawar, and Ryuki Tachibana. 2020. Bootstrapped q-learning with context relevant observation pruning to generalize in text-based games. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3002–3008.

[46] David Chen and Raymond Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 25.

[47] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 19:25–35.

[48] Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Wei Liu, and Tat-Seng Chua. 2017. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6298–6306. IEEE.

[49] Valerie Chen, Abhinav Gupta, and Kenneth Marino. 2021. Ask your humans: Using human instructions to improve generalization in reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[50] Xiaojun Chen, Shengbin Jia, and Yang Xiang. 2020. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948.

[51] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 97, pages 1282–1289.

[52] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Dominey, and Pierre-Yves Oudeyer. 2020. Language as a cognitive tool to imagine

goals in curiosity driven exploration. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33.

[53] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. *arXiv preprint arXiv:1806.11532*.

[54] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. 2018. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32.

[55] Abhishek Das, Federico Carnevale, Hamza Merzic, Laura Rimell, Rosalia Schneider, Josh Abramson, Alden Hung, Arun Ahuja, Stephen Clark, Greg Wayne, et al. 2020. Probing emergent semantics in predictive agents via question answering. In *International Conference on Machine Learning (ICML)*, pages 2376–2391. PMLR.

[56] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. 2018. Embodied question answering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2135–213509. IEEE.

[57] Peter Dayan and Geoffrey E Hinton. 1992. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 5.

[58] MP Deisenroth and CE Rasmussen. 2011. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 465–473.

[59] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186.

[60] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. 2019. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2694–2703.

[61] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Tom Griffiths, and Alexei Efros. 2018. Investigating human priors for playing video games. In *International Conference on Machine Learning (ICML)*, pages 1349–1357. PMLR.

[62] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

[63] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, pages 1–50.

[64] Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 958–967.

[65] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations (ICLR)*.

[66] Jerome Feldman and Srinivas Narayanan. 2004. Embodied meaning in a neural theory of language. *Brain and language*, 89:385–392.

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

[68] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. 2016. Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE.

[69] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, et al. 2018. Noisy networks for exploration. In *International Conference on Learning Representations (ICLR)*.

[70] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11:219–354.

[71] Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. 2019. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *International Conference on Learning Representations (ICLR)*.

[72] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pages 1587–1596. PMLR.

[73] Nancy Fulda, Daniel Ricks, Ben Murdoch, and David Wingate. 2017. What can you do with a rock? affordance extraction via word embeddings. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1039–1045.

[74] Zhe Gan, Yu Cheng, Ahmed Kholy, Linjie Li, Jingjing Liu, and Jianfeng Gao. 2019. Multi-step reasoning via recurrent dual attention for visual dialog. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6463–6474.

[75] Jonas Gehring, Gabriel Synnaeve, Andreas Krause, and Nicolas Usunier. 2021. Hierarchical skills for efficient exploration. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 11553–11564.

[76] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. 2019. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2385–2391.

[77] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. 2018. Visualizing and understanding atari agents. In *International Conference on Machine Learning (ICML)*, pages 1792–1801. PMLR.

[78] Xiaoxiao Guo, Mo Yu, Yupeng Gao, Chuang Gan, Murray Campbell, and Shiyu Chang. 2020. Interactive fiction game playing as multi-paragraph reading comprehension with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7755–7765.

[79] David Ha and Jürgen Schmidhuber. 2018. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31.

[80] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870.

[81] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[82] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*, pages 2555–2565.

[83] Steven Hansen, Will Dabney, Andre Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. 2019. Fast task inference with variational intrinsic successor features. In *International Conference on Learning Representations (ICLR)*.

[84] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 30, pages 2094–2100.

[85] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 7903–7910.

[86] Matthew Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D Williams. 2019. Nail: A general interactive fiction agent. *arXiv preprint arXiv:1902.04259*.

[87] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[88] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. Deep reinforcement learning with a natural language action space. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1621–1630.

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[90] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al. 2017. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

[91] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32.

[92] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32.

[93] Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. 2021. Grounded language learning fast and slow. In *International Conference on Learning Representations (ICLR)*.

[94] Brian Hlubocky and Eyal Amir. 2004. Knowledge-gathering agents in adventure games. In *AAAI Workshop on Challenges in Game AI*.

[95] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 4565–4573.

[96] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

[97] Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. 2019. Hierarchical decision making by generating and following natural language instructions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 10025–10034.

[98] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 804–813.

[99] Drew Arad Hudson and Christopher D. Manning. 2018. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations (ICLR)*.

[100] Jena D Hwang, Chandra Bhagavatula, Ronan Le Bras, Jeff Da, Keisuke Sakaguchi, Antoine Bosselut, and Yejin Choi. 2021. (comet-) atomic 2020: On symbolic and neural commonsense knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 6384–6392.

[101] Sarthak Jain and Byron C Wallace. 2019. Attention is not explanation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 3543–3556.

[102] Vishal Jain, William Fedus, Hugo Larochelle, Doina Precup, and Marc G Bellemare. 2020. Algorithmic improvements for deep reinforcement learning applied to interactive fiction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 4328–4336.

[103] Michael Janner, Karthik Narasimhan, and Regina Barzilay. 2018. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61.

[104] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514.

[105] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. 2019. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 9419–9431.

[106] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *IEEE Transactions on Games*, 12:1–20.

[107] Leslie Pack Kaelbling. 1993. Learning to achieve goals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1094–1098.

[108] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. 2020. Model based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*.

[109] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 267–274.

[110] Sham M Kakade. 2001. A natural policy gradient. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 14, pages 1531–1538.

[111] Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. 2019. Modeling the long term future in model-based reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[112] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

[113] Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. 2018. Bilinear attention networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1564–1574.

[114] Wonjae Kim and Yoonho Lee. 2019. Learning dynamics of attention: Human prior for interpretable machine reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6019–6030.

[115] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274.

[116] Ryosuke Kohita, Akifumi Wachi, Daiki Kimura, Subhajit Chaudhury, Michiaki Tatsubori, and Asim Munawar. 2021. Language-based general action template for reinforcement learning agents. In *Findings of the Association for Computational Linguistics (ACL-Findings)*, pages 2125–2139.

[117] Vijay R. Konda and John N. Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12, pages 1008–1014.

[118] Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*.

[119] Bartosz Kostka, Jaroslaw Kwiecieli, Jakub Kowalski, and Pawel Rychlikowski. 2017. Text-based adventures of the golovin ai agent. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 181–188. IEEE.

[120] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 29, pages 3675–3683.

[121] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40.

[122] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations (ICLR)*.

[123] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

[124] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1192–1202.

[125] Yuxi Li. 2019. Reinforcement learning applications. *arXiv preprint arXiv:1908.06973*.

[126] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[127] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Proceedings of the Conference on*

Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2822–2832.

[128] Long-Ji Lin. 1992. Self-improving reactive agents besed on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

[129] Grace W Lindsay. 2020. Attention in psychology, neuroscience, and machine learning. *Frontiers in Computational Neuroscience*, 14:29.

[130] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. 2021. From motor control to team play in simulated humanoid football. *arXiv preprint arXiv:2105.12196*.

[131] Yang Liu and Mirella Lapata. 2019. Hierarchical transformers for multi-document summarization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5070–5081.

[132] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. 2016. Hierarchical question-image co-attention for visual question answering. In *Advances in neural information processing systems (NeurIPS)*, pages 289–297.

[133] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. 2019. A survey of reinforcement learning informed by natural language. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 6309–6317.

[134] James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. 2015. Grounding english commands to reward functions. In *Robotics: Science and Systems (RSS)*.

[135] Andrea Madotto, Mahdi Namazifar, Joost Huizinga, Piero Molino, Adrien Ecoffet, Huaixiu Zheng, Dian Yu, Alexandros Papangelis, Chandra Khatri, and Gokhan Tur. 2020. Exploration based language learning for text-based games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1488–1494.

[136] Christopher Manning and Hinrich Schutze. 1999. *Foundations of statistical natural language processing*. MIT Press.

[137] David Mascharka, Philip Tran, Ryan Soklaski, and Arjun Majumdar. 2018. Transparency by design: Closing the gap between performance and interpretability in visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4942–4950.

[138] Steve Meretzky. 1983. Planetfall. Infocom.

[139] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC)*.

[140] Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. 2021. Ella: Exploration through learned language abstraction. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 29529–29540.

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

[142] Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1004–1015.

[143] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 48, pages 1928–1937.

[144] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[145] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[146] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. 2019. Towards interpretable reinforcement learning using attention augmented agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 12350–12359.

[147] Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. 2021. Text-based rl agents with commonsense knowledge: New challenges, environments and baselines. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 9018–9027.

[148] Keerthiram Murugesan, Mattia Atzeni, Pushkar Shukla, Mrinmaya Sachan, Pavan Kapanipathi, and Kartik Talamadupula. 2020. Enhancing text-based reinforcement learning agents with commonsense knowledge. *arXiv preprint arXiv:2005.00811*.

[149] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. 2018. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 3303–3313.

[150] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. 2018. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9191–9200.

[151] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. 2018. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874.

[152] Karthik Narasimhan, Tejas D Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–11.

[153] Andrew Y Ng and Stuart J Russell. 2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 663–670.

[154] Montfort Nick. 2005. *Twisty Little Passages: An Approach to Interactive Fiction*. MIT Press.

[155] William Noble and Iain Davidson. 1996. *Human Evolution, Language and Mind: A Psychological and Archaeological Inquiry*. CUP Archive.

[156] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. 2015. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2863–2871.

[157] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. 2017. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pages 2661–2670.

[158] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. 2017. Count-based exploration with neural density models. In *International Conference on Machine Learning (ICML)*, pages 2721–2730. PMLR.

[159] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. 2020. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 7487–7498. PMLR.

[160] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, pages 2778–2787.

[161] Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. 2017. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2231–2240.

[162] Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. 2020. Skew-fit: State-covering self-supervised reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 7783–7792. PMLR.

[163] Sébastien Racanière, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. 2017. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5694–5705.

[164] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*.

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

[166] Dhanesh Ramachandram and Graham W Taylor. 2017. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine*, 34(6):96–108.

[167] Siddharth Reddy, Anca D Dragan, and Sergey Levine. 2019. Sqil: Imitation learning via reinforcement learning with sparse rewards. In *International Conference on Learning Representations (ICLR)*.

[168] Jean-Jacques Rousseau and Johann Gottfried Herder. 2012. *On the Origin of Language*. University of Chicago Press.

[169] Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M Lake. 2020. A benchmark for systematic generalization in grounded language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33.

[170] Abdelrhman Saleh, Natasha Jaques, Asma Ghandeharioun, Judy Shen, and Rosalind Picard. 2020. Hierarchical reinforcement learning for open-domain dialog. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 8741–8748.

[171] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

[172] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*.

[173] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference (ESWC)*, pages 593–607.

[174] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897.

[175] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*.

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

[177] Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, R Devon Hjelm, Philip Bachman, and Aaron C Courville. 2021. Pretraining representations for data-efficient reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 12686–12699.

[178] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. 2019. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*.

[179] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. 2018. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning (ICML)*, volume 80, pages 4654–4663.

[180] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*.

[181] Tianmin Shu, Caiming Xiong, and Richard Socher. 2018. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[182] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[183] Ishika Singh, Gargi Singh, and Ashutosh Modi. 2021. Pre-trained language models as prior knowledge for playing text-based games. *arXiv preprint arXiv:2107.08408*.

[184] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. 2018. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 7156–7166.

[185] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 31.

[186] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2018. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning (ICML)*, pages 4732–4741.

[187] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, page 2377–2385.

[188] Pei-Hao Su, Paweł Budzianowski, Stefan Ultes, Milica Gasic, and Steve Young. 2017. Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 147–157.

[189] Richard S Sutton. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2:160–163.

[190] Richard S Sutton and Andrew G Barto. 1998. *Introduction to Reinforcement Learning*. MIT Press.

[191] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

[192] Erik Talvitie. 2014. Model regularization for stable sample rollouts. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 780–789.

[193] Ruo Yu Tao, Marc-Alexandre Côté, Xingdi Yuan, and Layla El Asri. 2018. Towards solving text-based games by producing adaptive action spaces. *arXiv preprint arXiv:1812.00855*.

[194] Guy Tennenholtz and Shie Mannor. 2019. The natural language of actions. In *International Conference on Machine Learning (ICML)*, pages 6196–6205.

[195] Pedro A Tsividis, Thomas Pouncy, Jaqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Human learning in atari. In *2017 AAAI Spring Symposium Series*.

[196] Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. 2019. Learning to speak and act in a fantasy text adventure game. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 673–683.

[197] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5998–6008.

[198] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

[199] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 70, pages 3540–3549.

[200] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[201] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Dkn: Deep knowledge-aware network for news recommendation. In *Proceedings of the World Wide Web Conference (WWW)*, page 1835–1844.

[202] Sida I Wang, Percy Liang, and Christopher D Manning. 2016. Learning language games through interaction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2368–2378.

[203] Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. 2019. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 5329–5336.

[204] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *Proceedings of the World Wide Web Conference (WWW)*, pages 2022–2032.

[205] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. 2019. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6629–6638.

[206] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. 2018. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 37–53.

[207] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning (ICML)*, pages 1995–2003. PMLR.

[208] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning*, 8:279–292.

[209] Sam Wenke, Dan Saunders, Mike Qiu, and Jim Fleming. 2019. Reasoning and generalization in rl: A tool use perspective. *arXiv preprint arXiv:1907.02050*.

[210] Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20.

[211] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

[212] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5285–5294.

[213] Yikun Xian, Zuohui Fu, S. Muthukrishnan, Gerard de Melo, and Yongfeng Zhang. 2019. Reinforcement knowledge graph reasoning for explainable recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, page 285–294.

[214] Yunqiu Xu, Ling Chen, Meng Fang, Yang Wang, and Chengqi Zhang. 2020. Deep reinforcement learning with transformers for text adventure games. In *IEEE Conference on Games (CoG)*, pages 65–72.

[215] Yunqiu Xu, Meng Fang, Ling Chen, Yali Du, and Chengqi Zhang. 2021. Generalization in text-based games via hierarchical reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP-Findings)*, pages 1343–1353.

[216] Yunqiu Xu, Meng Fang, Ling Chen, Yali Du, Joey Tianyi Zhou, and Chengqi Zhang. 2020. Deep reinforcement learning with stacked hierarchical attention for text-based games. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 16495–16507.

[217] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. 2016. Stacked attention networks for image question answering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21–29. IEEE.

[218] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1480–1489.

[219] Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep CALM and explore: Language models for action generation in text-based games. In *Proceedings of*

the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 8736–8754.

[220] Xusen Yin and Jonathan May. 2019. Comprehensible context-driven text game playing. *IEEE Conference on Games (CoG)*, pages 1–8.

[221] Xusen Yin, Ralph Weischedel, and Jonathan May. 2020. Learning to generalize for sequential decision making. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Findings (EMNLP-Findings)*, pages 3046–3063.

[222] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 5812–5823.

[223] Adams Wei Yu, David Dohan, Quoc Le, Thang Luong, Rui Zhao, and Kai Chen. 2018. Fast and accurate reading comprehension by combining self-attention and convolution. In *International Conference on Learning Representations (ICLR)*.

[224] Mo Yu, Xiaoxiao Guo, Yufei Feng, Xiaodan Zhu, Michael Greenspan, and Murray Campbell. 2020. Deriving commonsense inference tasks from interactive fictions. *arXiv preprint arXiv:2010.09788*.

[225] Yunlong Yu, Zhong Ji, Yanwei Fu, Jichang Guo, Yanwei Pang, and Zhongfei (Mark) Zhang. 2018. Stacked semantics-guided attention model for fine-grained zero-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 5995–6004.

[226] Xingdi (Eric) Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. 2018. Counting to explore and generalize in text-based games. In *European Workshop on Reinforcement Learning (EWRL)*.

[227] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 3562–3573.

[228] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. 2019. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations (ICLR)*.

[229] Mikuláš Zelinka. 2018. Baselines for reinforcement learning in text games. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 320–327. IEEE.

[230] Mikulas Zelinka, Xingdi Yuan, Marc-Alexandre Côté, Romain Laroche, and Adam Trischler. 2019. Building dynamic knowledge graphs from text-based games. *arXiv preprint arXiv:1910.09532*.

[231] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 793–803.

[232] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 353–362.

[233] Min-Ling Zhang and Zhi-Hua Zhou. 2013. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

[234] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. 2020. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*.

[235] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32.

[236] Zhao Zhang, Fuzhen Zhuang, Meng Qu, Fen Lin, and Qing He. 2018. Knowledge graph embedding with hierarchical relation structure. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3198–3207.

[237] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 11015–11023.

[238] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. Deep reinforcement learning for search, recommendation, and online advertising: A survey. *ACM SIGWEB Newsletter*, pages 1–15.

[239] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. 2019. Rtfm: Generalising to new environment dynamics via reading. In *International Conference on Learning Representations (ICLR)*.

[240] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. 2018. Reinforcement and imitation learning for diverse visuomotor skills. In *Proceedings of Robotics: Science and Systems (RSS)*.