ScriptWorld: Text Based Environment For Learning Procedural Knowledge

Abhinav Joshi , **Areeb Ahmad** , **Umang Pandey** , **Ashutosh Modi**
Indian Institute of Technology Kanpur (IIT-K)
{ajoshi, ashutoshm}@cse.iitk.ac.in, {areeb, umangp}@iitk.ac.in

Abstract

Text-based games provide a framework for developing natural language understanding and commonsense knowledge about the world in reinforcement learning based agents. Existing text-based environments often rely on fictional situations and characters to create a gaming framework and are far from real-world scenarios. In this paper, we introduce `ScriptWorld`: a text-based environment for teaching agents about real-world daily chores and hence imparting commonsense knowledge. To the best of our knowledge, it is the first interactive text-based gaming framework that consists of daily real-world human activities designed using scripts dataset. We provide gaming environments for 10 daily activities and perform a detailed analysis of the proposed environment. We develop RL-based baseline models/agents to play the games in `ScriptWorld`. To understand the role of language models in such environments, we leverage features obtained from pre-trained language models in the RL agents. Our experiments show that prior knowledge obtained from a pre-trained language model helps to solve real-world text-based gaming environments.

1 Introduction

Text-based games in reinforcement learning have attracted research interests in recent years [Hausknecht *et al.*, 2020; Küttler *et al.*, 2020]. These games have been developed to impart Natural Language Understanding (NLU) and commonsense reasoning capabilities in Reinforcement Learning (RL) based agents. A typical text-based game consists of a textual description of states of an environment where the agent/player observes and understands the game state and context using text and interacts with the environment using textual commands (actions). For successfully solving a text-based game, in addition to language understanding, an agent needs complex decision-making abilities, memory, planning, questioning, and commonsense knowledge [Côté *et al.*, 2018]. Existing text-based gaming frameworks (e.g., Jericho [Hausknecht *et al.*, 2020]) provide a rich fictional setup (e.g., treasure hunt in a fantasy world) and require an agent to take complex decisions involving language and fantasy world knowledge. However, the existing text-based frameworks are created using a fixed prototype and are often distant from real-world scenarios involving daily human activities. Though these frameworks aim to provide a rich training bench for enhancing NLU in RL algorithms, the fictional concepts in these games are not well grounded in real-world scenarios, making the learned knowledge non-applicable to the real world. In contrast, for trained RL algorithms to be of practical utility, they should be trained in real-world scenarios that involve daily human activities. Humans carry out daily activities (e.g., making coffee, going for a bath) without much effort by making use of implicit *Script Knowledge*.

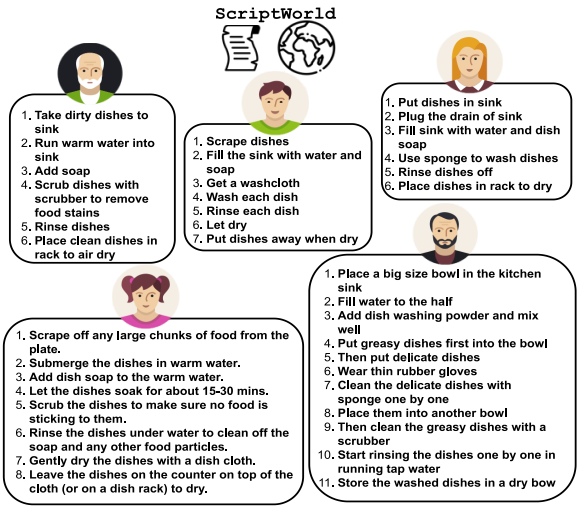

Figure 1: Different descriptions for the `Washing Dishes` script scenario.

Formally, **Scripts** are defined as sequences of actions describing stereotypical human activities, for example, cooking pasta, making coffee, etc. [Schank and Abelson, 1975]. Scripts entail knowledge about the world. For example, when someone talks about "Washing Dishes", there lies an implicit knowledge of fine-grained steps which would be present in the activity. By just saying, "I washed dishes on Thursday," a person conveys the implicit knowledge about the entire process (Fig. 1). The detailed implicit understanding of a

task not only helps to learn about an activity but also facilitates taking suitable actions depending on the environment and past choices. Moreover, for learning a new task, humans can quickly and effortlessly discover new skills for performing the task either by their knowledge about the world or by reading (a manual) about it. With the aim to promote similar learning behavior in RL agents, in this paper, we propose **ScriptWorld**, a new text-based game environment based on real-world scenarios involving script knowledge.

The motivation for creating ScriptWorld environment is threefold. Firstly, ScriptWorld environment is based on the concept of scripts that encapsulates commonsense and procedural knowledge about the world. The environment is designed to enable agents to learn this knowledge while participating in the game. Scripts have non-linear structure [Wanzare *et al.*, 2016]. A script scenario can be described in multiple ways with linguistic variation across different descriptions. Fig. 1 shows different descriptions for the washing dishes scenario. Moreover, at the level of execution, the order of events/actions within the script can vary across different descriptions of a scenario. For example, some events may be skipped, and the order of events might vary. Hence, learning script knowledge is challenging. Taking into account the variability in descriptions of a scenario, an agent needs to learn the prototypical order of events and needs to abstract out the meaning of different verbal descriptions of an action. Secondly, ScriptWorld being a text-based environment about everyday scenarios, provides an opportunity for grounded language learning and understanding. Language phenomena do not happen in isolation, but the semantics are grounded in the real world [Hill *et al.*, 2017]; ScriptWorld provides the environment to establish and learn that grounding. Lastly, there have been extensive studies that have explored the cognitive basis of script knowledge in humans [Miikkulainen and Elman, 1993; Modi, 2017]. ScriptWorld involves the acquisition of script knowledge. Consequently, it provides an opportunity to compare the behavior of a trained RL agent with humans providing further insights into the cognitive aspects.

In a nutshell, we make the following contributions:

- We introduce a new interactive text-based gaming environment, ScriptWorldconsisting of games based on script descriptions provided by human annotators for performing realistic daily chores. We perform a detailed analysis of the proposed environment and release the environment and agents: https://github.com/Exploration-Lab/ScriptWorld.
- We propose and experiment with a battery of Reinforcement Learning (RL) agents based on pre-trained Language Models (LM) as baselines for solving the ScriptWorld environment. The experiments show that pre-trained LMs, when combined with RL agents, give reasonable performance, pointing towards scope for improvement and inclusion of prior knowledge.

2 Related Work

Text Based Games. Text-based games are divided into three main categories based on how an agent/player might issue (take) commands (actions): Parser-based, Choice Base, and Hyper Text Based [He *et al.*, 2016]. The player issues a command in Parser-based games by typing in the input, and an inbuilt parser parses it. In Hypertext-based games, the player issues a command by selecting one of the Hyperlinks present in the prompt. In choice-based games, the player chooses the command from a list of options in addition to the state description. Parser-based games are limited since these can only parse sentences that adhere to predefined grammar and vocabulary. Giving flexibility for free-form text suffers from the exponentially increasing action space. ScriptWorld uses choice-based approach (also see §6). Moreover, in general, choice-based games are more popular among humans than parser-based games [He *et al.*, 2016]. Côté *et al.*, 2018 have introduced TextWorld sandbox environment, a Python-based framework in which the user can build parser-based game worlds of varying difficulty along with in-game objects and goal states while monitoring states and assigning rewards. Language diversity and complexity of action space are limited in TextWorld. In contrast, ScriptWorld (created using human written texts) overcomes these issues by generating ample alternative pathways to complete a task. The complexity and variability in ScriptWorld help to develop better language understanding capabilities in agents. Other Text-based game frameworks have been proposed, such as TWC (TextWorld Commonsense) [Murugesan *et al.*, 2020], and Question Answering with Interactive Text (QAit) [Yuan *et al.*, 2019] build on TextWorld. Similarly, Hausknecht *et al.*, 2020 have introduced a new framework called Jericho, which facilitates using man-made Interactive Fiction Games as learning environments for agents to train and learn.

Scripts. Scripts have been an active area of research for the last four decades. As evident from the definition (§1), scripts encapsulate commonsense and procedural knowledge about the world and hence are an ideal source for training agents to learn about the world. Several computational models have developed for modeling script knowledge, inter alia, [Regneri *et al.*, 2010; Frermann *et al.*, 2014; Modi, 2016; Modi and Titov, 2014; Rudinger *et al.*, 2015; Jans *et al.*, 2012; Pichotta and Mooney, 2016; Modi *et al.*, 2017]. A number of corpora have also been created, e.g., InScript [Modi *et al.*, 2016], DeScript [Wanzare *et al.*, 2016], McScript [Ostermann *et al.*, 2018a; Ostermann *et al.*, 2018b], and ProScript [Sakaguchi *et al.*, 2021]. Researchers have also examined script knowledge from the perspective of language modeling [Sancheti and Rudinger, 2022].

RL Agents. Narasimhan *et al.*, 2015 have introduced an RL-based architecture called LSTM-DQN that learns the action policies and state representations of parser-based games. A number of other agents have been proposed for text-based environments, e.g., He *et al.*, 2016 have introduced DRRN (Deep Reinforcement Relevance Network) architecture, KG-DQN architecture [Ammanabrolu and Riedl, 2019; Ammanabrolu and Hausknecht, 2020; Adhikari *et al.*, 2020; Chaudhury *et al.*, 2020; Adolphs and Hofmann, 2020; Yin and May, 2019; Yao *et al.*, 2020]. Singh *et al.*, 2022 introduce a pretrained language model finetuned on the dynamics

Scenario	Nodes	Deg.	Paths
Taking a **Bath**	525	3.7	$3.1e+27$
Baking a **Cake**	542	3.6	$4.0e+26$
Flying in an **Airplane**	528	3.6	$2.6e+30$
Going Grocery **Shopping**	544	3.7	$2.3e+26$
Going on a **Train**	427	3.7	$3.1e+21$
Planting a **Tree**	373	3.7	$1.6e+16$
Riding on a **Bus**	376	3.8	$1.0e+17$
Repairing Flat **Bicycle** Tire	402	3.4	$8.4e+18$
Borrowing Book from **Library**	397	3.7	$3.1e+19$
Getting a **Haircut**	528	3.7	$4.0e+28$

Table 1: The table compares graphs of different scenarios present in ScriptWorld. Deg. represents the average degree for the nodes in the scenario graph.

of the game to equip the agent with language learning capabilities as well as acquire real-world knowledge. Our baseline agents come close to Singh *et al.*, 2022.

3 ScriptWorld Environment

ScriptWorld tries to bridge the gap between real-world scenarios (via Scripts) and text-based games for RL by creating a suitable environment. We take into consideration three design choices for developing the environment: **1) Complexity:** The game environment should be complex enough to test an RL algorithm's capacity to capture, understand and remember reasonable steps required for performing a daily chore. **2) Flexibility:** For an environment to help develop and debug RL algorithms, it becomes imperative to consider flexibility as a feature. The environment should be flexible regarding difficulty levels and handicaps (hints) to provide a good test bench for reinforcement learning algorithms. **3) Relation to Real-World scenarios:** The environment should consist of activities/tasks grounded in the real world and well understood among humans.

DeScript. Given the nature of Script knowledge, we use a scripts corpus referred to as DeScript [Wanzare *et al.*, 2016] for creating ScriptWorld environment. DeScript is a corpus having a telegram-style sequential description of a scenario in English (e.g., baking a cake, taking a bath, etc.) DeScript is created via crowd-sourcing. For a given scenario, crowd-workers write a point-wise and sequential short description of various events involved in executing the scenario (this one complete description is called an ESD (Event Sequence Description)). Fig 1 shows an example of 5 ESDs for the Washing Dishes scenario. DeScript collects data for 40 daily activities (scenarios), and 100 ESDs (written by different crowd-sourced workers) are collected for each scenario. Additionally, for a given scenario, semantically similar events from different ESDs are manually aligned by human annotators (more details about data collection and annotations are present in Wanzare *et al.*, 2016). The alignment annotation is done for 10 scenarios (Table 1 gives the list of scenarios). In the present version of ScriptWorld, we only include these 10 scenarios with gold alignment. Another line of work can be to consider sequence alignment algorithms [Chatzou *et al.*, 2016] to align sequences for the remaining 30 scenarios. However, as observed in initial experiments, the error rate of alignment algorithms gets propagated to the graph formation leading to a less reliable environment. We leave the automatic alignment of the remaining 30 scenarios for future work. The gold alignments in the DeScsript corpus contain cluster annotations of similar events across multiple ESDs into a single abstract, generalized event. For example, Fig. 2 depicts the scenario, Get Medicine, where similar events from ESDs written by different people are clustered to form generalized event categories. Further, the combined set of events and the relation between the ESDs is leveraged to construct a graph (as explained later) where each node represents an abstract event. To the best of our knowledge, the proposed method is the first novel approach to create an environment (based on script knowledge) that could be useful for training RL agents.

The ScriptWorld environment is created from scratch using Python. A typical game begins by providing a quest (goal) to the agent. The quest/goal is a one-line description of the scenario (e.g., plant a tree). The agent is also provided with initial observations (in English). Since it is a choice-based game, at each step in the game, the agent is also presented with a list of actions/choices (in English) that it could opt to advance towards the goal. Based on the action selected by the agent, it is awarded a zero/positive/negative reward at each step. Every correct action takes the agent closer to task completion, whereas every wrong action results in a deviated path. (also see App. A, the appendix is available at https://github.com/Exploration-Lab/ScriptWorld).

Graph Formation. DeScript provides set of aligned ESDs ($\mathcal{E}_1^{\mathcal{S}_i}, \mathcal{E}_2^{\mathcal{S}_i}, \ldots, \mathcal{E}_N^{\mathcal{S}_i}$) for a scenario $\mathcal{S}_i$. Each ESD $\mathcal{E}_k^i$ consists of sequence of short event descriptions: $\mathbf{e}_1^{(\mathcal{E}_k^i)}, \mathbf{e}_2^{(\mathcal{E}_k^i)}, \ldots \mathbf{e}_n^{(\mathcal{E}_k^i)}$. Gold alignment in DeScript results in events in different ESDs that are semantically similar, getting linked to each other, i.e., clustered together. For example, for the Washing Dishes scenario, events "put dishes in sink" ($\mathbf{e}_1^{(\mathcal{E}_1^{Wash})}$) in $\mathcal{E}_1^{Wash}$ and "take dirty dishes to sink" ($\mathbf{e}_1^{(\mathcal{E}_2^{Wash})}$) in $\mathcal{E}_2^{Wash}$ are linked (clustered) together. Aligned events (from different ESDs) are used to create a graph having nodes as the event clusters (of aligned events) and directed edges representing the prototypical order of the events. In particular, a directed edge is drawn from node p to q if there is at least one event in node p that directly precedes an event in node q. We refer to the created event node graph as the *compact graph* (Fig. 3), compact graphs for other scenarios are in App. A. The alignment annotations in the DeScript also group multiple sets of actions that belong to the same event. For example, an event "go to the terrace" can be performed in two sets of sequenced steps by different annotators. 1) call the elevator $\rightarrow$ step in elevator $\rightarrow$ step out at the top floor, and 2) find stairs $\rightarrow$ climb stairs $\rightarrow$ reach top floor. We leverage the presence of such instances in the graph node to enrich the complexity of our environment. We split each event node in the compact graph into two nodes, the entry event node and the exit event node. Further, multiple action sequences result in parallel paths for reaching the exit node from the entry node (see also App. A). For instance, the above example will result in two parallel paths, where a player or an agent has to decide at the entry node to either take the elevator or the stairs. If players choose to take the stairs, they are expected to

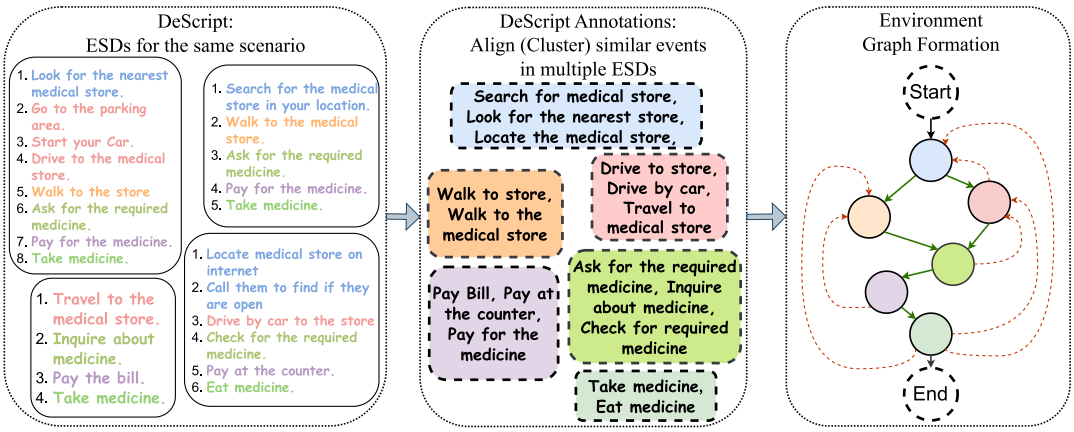

Figure 2: The figure shows a simplified version of the scenario, `Get Medicine`, and the process of creating an environment graph (right diag.) from the ESDs (left diag.) and aligned events (middle diag.) for the scenario. The green directed edges in the environment graph represent the correct paths, and the red edges denote the environment transition when a wrong option is selected.

follow the next set of actions to reach the terrace. Moreover, all the sub-steps in this event now result in multiple graph nodes. We refer to this graph as the *scenario graph* (see App. A). This helps to capture the variability in performing daily chores, making the environment more realistic. Though the DeScript corpus provides clustered events for every scenario, after graph creation, we found that a few of the ESDs present in the corpus were inconsistent, not fitting the commonsense reasoning for a procedure. We also observed that some of the ESDs written by annotators are too small and describe the task in generic terms. Such ESDs, when considered in graph formation, result in direct paths to the final goal node, making the game less complex. We remove all such inconsistencies from the graph by manual inspection, making it more reliable for capturing script knowledge and keeping the realism intact for the environment. The compact graph serves as an initial starting point for creating the scenario graph. The agents are trained on a scenario graph.

To quantitatively capture the complexity of scenarios in `ScriptWorld`, we calculate the total number of paths reaching the end node from the start node. We first compute the total number of paths in the compact graph using a depth-first traversal. Further, we extend the computation by adding the number of parallel paths present for each entry and exit event node in the scenario graph. TotalPaths $= \sum_{p_k=0}^{T} \prod_{i=1}^{N} {n_i}^{(p_k)}$, where T is the total number of paths in a compact graph, N represents the total number of nodes in a path p_k and ${n_i}^{(p_k)}$ denotes the number of splits for the i^{th} node. Table 1 shows the total number of paths. As evident from the table, the number of paths in each of the scenarios is enormous and demonstrates the highly complex nature of the environment. Overall, the scenario `Flying in an Airplane` turns out to be the most complex one in terms of the number of correct possible paths. This is possibly due to more variability in carrying out this activity.

Environment Creation. We create the game environment using scenario graphs. For each state in the environment, the agent is required to pick the correct action (choice) from the available options. Since the created scenario graph contains a wide variety of suitable actions grouped in a node, we sample the right choice from the available actions in a node. Note that sampling of correct actions happens randomly at every visit, making the environment highly dynamic. To create incorrect choices, we exploit the temporal nature of the scenario graphs. As a scenario graph contains the sequence of actions to perform a specific sub-task, all actions in nodes (both past as well as future nodes are considered) that are far from the current node become invalid for the current state. For selecting this node distance, we manually experiment with different node distances and find the different distances $(d_1, d_2, \ldots d_{10})$ suitable for sampling the invalid actions, i.e., for a scenario i, we consider all nodes at a distance greater than d_i hops from the current node (Table in App. A shows various distances chosen for each of the scenarios). This strategy of sampling the invalid choices makes the environment more complex as all the options are related to the same scenario, and an understanding of event order in a task is required to achieve the goal.

Rewards (Performance Scores): For all the scenarios, every incorrect action choice results in a negative reward of -1, and every correct choice returns a 0 reward. For task completion, the agent gets a reward of 10, i.e., a player gets a maximum reward of 10 at the end of each game if they choose a correct sequence of actions. The choice of zero rewards for correct action helps RL algorithms explore multiple correct ways of performing a task, capturing the generalized procedural knowledge required for a specific task. The game terminates when an agent chooses 5 successive wrong actions.

Flexibility: To introduce flexibility in `ScriptWorld`, we consider two settings in a game. **1) Number of choices:** At each step, the number of choices presented to an agent can be changed (1 correct choice and the rest all incorrect). As the number of options increases, it becomes more challenging for an agent to choose the right action. **2) Number of backward hops for wrong actions:** We choose the number of backward

Algorithm	DQN		A2C		PPO		RPPO	
	handicap	w/o handicap	handicap	w/o handicap	handicap	w/o handicap	handicap	w/o handicap
Shopping	9.60 (± 0.62)	-7.28 (± 13.15)	9.90 (± 0.30)	-9.81 (± 14.71)	9.84 (± 0.39)	-4.78 (± 10.79)	9.71 (± 0.57)	**8.79 (± 4.15)**
Bus	8.98 (± 0.79)	-1.47 (± 11.16)	9.89 (± 0.34)	-7.37 (± 17.09)	9.93 (± 0.25)	1.50 (± 7.50)	9.97 (± 0.17)	**9.32 (± 1.24)**
Train	9.21 (± 2.07)	-3.10 (± 11.16)	9.89 (± 0.31)	-8.13 (± 14.99)	9.75 (± 0.49)	-1.13 (± 9.47)	9.56 (± 0.80)	**8.19 (± 4.70)**
Library	9.51 (± 0.68)	-1.94 (± 9.87)	9.88 (± 0.32)	-3.03 (± 9.84)	9.90 (± 0.30)	1.12 (± 7.31)	9.89 (± 0.31)	**8.41 (± 4.77)**
Haircut	9.88 (± 0.35)	-9.30 (± 12.93)	9.89 (± 0.34)	-5.87 (± 12.28)	9.85 (± 0.38)	-4.30 (± 10.84)	9.63 (± 0.64)	**6.32 (± 5.29)**
Cake	9.32 (± 0.84)	-4.13 (± 9.22)	9.48 (± 0.92)	-7.58 (± 13.18)	9.87 (± 0.34)	-4.46 (± 12.32)	9.78 (± 0.48)	**7.18 (± 4.97)**
Bicycle	9.50 (± 0.75)	0.07 (± 7.89)	9.95 (± 0.22)	-3.49 (± 12.39)	9.90 (± 0.33)	1.17 (± 6.93)	9.74 (± 0.57)	**7.85 (± 5.12)**
Tree	9.94 (± 0.24)	-0.15 (± 7.83)	9.86 (± 0.44)	-3.54 (± 12.56)	9.98 (± 0.14)	1.43 (± 7.29)	9.96 (± 0.19)	**8.88 (± 3.23)**
Airplane	9.68 (± 0.75)	-4.21 (± 12.39)	9.86 (± 0.35)	-8.66 (± 12.66)	9.86 (± 0.40)	-4.74 (± 11.08)	9.54 (± 0.73)	**6.85 (± 6.12)**
Bath	9.68 (± 0.61)	-6.49 (± 13.23)	9.75 (± 0.57)	-10.02 (± 15.95)	9.84 (± 0.37)	-5.35 (± 11.19)	9.45 (± 0.82)	**6.35 (± 5.59)**

Table 2: The table shows performance scores (averaged over multiple runs) of various agents for all the scenarios (number of choices = 2). The number in brackets shows the standard deviation of the score. Paraphrase Albert Small V2 is used as the LM

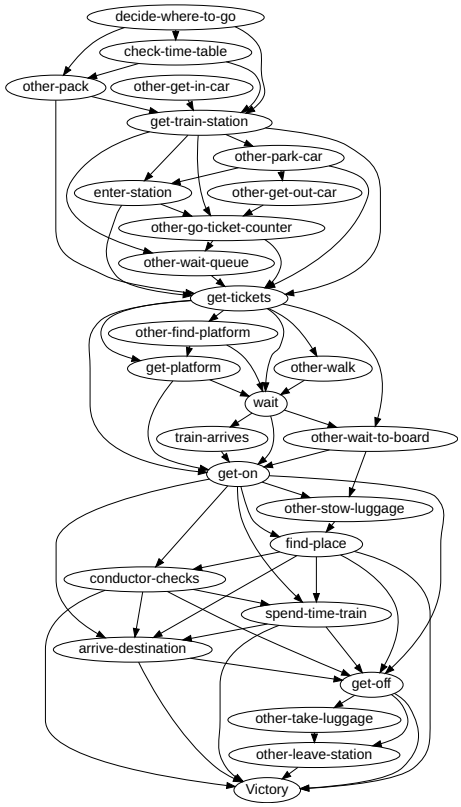

Figure 3: The figure shows the *"compact graph"* created for the scenario `Going on a Train`.

hops as another game setting that decides how many hops to displace whenever a wrong action is selected. When an agent selects an incorrect choice, its location is displaced by hopping it backward in the temporal domain, and this back-hop distance is another parameter in the environment. In our experiments, agents played with the environment with a back-hop distance of 1. Due to the presence of parallel paths in the graph, an agent hops to a previous node in case of incorrect action and may not follow the same path again, which acts as a penalty. For the start node, since backward hop is not possible, the agent remains at the same position; however, both positive and negative choices are re-sampled, and consequently, observations change. These parameters introduce flexibility in our environment, giving the freedom to create a suitable test bench for RL algorithms.

Handicaps (Hints): Text-based games are often challenging for RL agents playing from scratch. To mitigate the complexity issue, we introduce a version of the game with hints (referred to as handicaps) for each state. The hint for a state provides a short textual clue for the next action to take at the current state. The presence of hints in the environment makes the gameplay relatively easier. Hints are generated automatically using GPT2 [Radford *et al.*, 2019]. Scenario title concatenated with state node event description (separated by a full-stop) is given as the prompt to GPT2 for generating a large number of hints, and then a hint is sampled from them. We manually examined the hints to ensure they did not repeat (verbatim) any of the existing actions. To introduce variability, one could also stochastically decide to show a hint, e.g., by sampling from a Bernoulli distribution at each state. However, in this paper, we consider only the setting where hints are shown at every state. We leave this for future work.

Comparison with other text-based environments: `ScriptWorld` environment is different from the existing text-world-based environments (e.g., Text World, Jericho, TWC, QAit). The primary novelty of `ScriptWorld` comes from the inclusion of realistic scenarios made by leveraging ESDs written by human annotators, and this requires procedural knowledge to solve the game. The complexity (Table 1) of the `ScriptWorld` is much more than the existing environments, requiring the agent to remember past events and actions. We provide more details about `ScriptWorld` and compare it with other environments in App. A.

4 RL Baselines

In the `ScriptWorld` environment, for every state, the environment returns a sample of a possible set of choices. Since these choices provide feedback related to the current state, the agent must keep track of all the observations received after a particular choice. This property typically resembles the Partially Observable Markov decision processes (POMDP) [Kaelbling *et al.*, 1998], where the agent can never observe the complete state of the environment. Formally, `ScriptWorld` is defined by $(S, A, \Omega, R, \gamma)$, where S is the set of environment states (nodes in the scenario graph), and A is the set of all actions (choices), Ω is the set of observations, i.e., description of various actions, R is the reward obtained and γ is the discount parameter. The goal of an agent is to learn a policy $\pi(a \mid s)$, i.e., a mapping from a set of observations to actions leading to an optimal choice in a particular

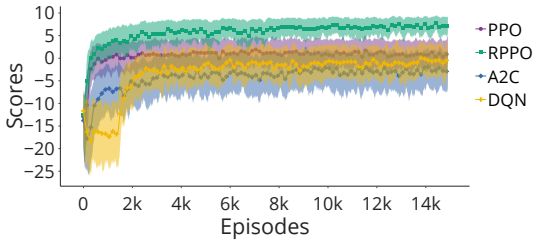

Figure 4: The figure shows the performance comparison of multiple RL algorithms on scenario `Repairing a Flat Bicycle Tire` on setting (without handicap, choices=2). Paraphrase Albert Small V2 is used as the LM. The plot shows moving average of performance curves across various episodes

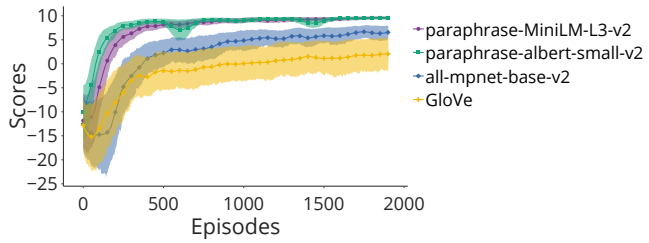

Figure 5: The figure shows the performance of the RPPO algorithm with various language models on scenario `Repairing a Flat Bicycle Tire` on setting (with handicap, choices=2). (highlighting the importance of LMs (contextual embeddings) over GloVe (non-contextual)).

state. In some algorithms (e.g., DQN: Deep Q-Network), instead of learning the policy, the agent learns q-values, which can reveal the policy. Formally, q-value (q-function) $Q_\pi(s, a)$ is the expected cumulative return if an agent starts from state s and takes action a and thereafter follows a policy π. Recent developments in RL have proposed an approximation of $\pi(a \mid s)$/q-value via a parameterized model that takes state (features) and actions (features) as input and produces the $\pi(a \mid s)$/q-value as the output [Sutton and Barto, 2018]. We follow the same approach.

Recently, Language Models (LM) have shown promising results in almost all tasks in NLP (e.g., [Sancheti and Rudinger, 2022]). For the RL baselines for the `ScriptWorld` environment, we consider using pre-trained SBERT language models [Reimers and Gurevych, 2019] as a source of prior real-world knowledge, which could be used directly by an RL algorithm to solve the environment. We consider a generalized scheme where a pre-trained language model extracts information from observations, i.e., the features extracted ($h_i = \text{LM}(c_i)$) from the available set of choices $c \in \{c_1, \ldots, c_n\}$) is used by the RL algorithms as input features. The pre-trained language model generates embeddings (h_i) corresponding to each of the provided n options. The obtained embeddings are concatenated (O) and passed as input to the RL algorithm, i.e.,

$$c \in \{c_1, \ldots, c_n\}; \quad h_i = \text{LM}(c_i)$$
$$O = h_1 \oplus h_2 \oplus \ldots \oplus h_n$$

Subsequently, the RL framework generates $\pi(a \mid s)$/Q values for the available set of actions. With the help of this generalized architecture, we run a detailed set of experiments with combinations of multiple language models and different RL algorithms. In particular, we use DQN [Mnih *et al.*, 2013], A2C [Mnih *et al.*, 2016], PPO [Schulman *et al.*, 2017], and RPPO: Recurrent PPO (PPO + LSTM). More details about RL agents, training, and other settings are provided in App. B. Some of the other existing works for language-based RL algorithms use knowledge-based agents. As these KBs do not directly adapt to our setting, we could not experiment with these approaches. In the future, we would explore how to make use of external knowledge to incorporate into the agent.

5 Experiments, Results and Analysis

RL Agents Performances: To benchmark the performance of existing RL algorithms on `ScriptWorld` we perform an extensive set of experiments considering various combinations of language model embeddings and popular RL algorithms. Due to space limitations, we report the primary findings here, and the remaining are discussed in the App. D. Table 2 shows the performance of various RL algorithms in all the scenarios. The performance score is the score (total reward) achieved by an agent till the point of termination. As `ScriptWorld` was designed, keeping flexibility the primary feature, in Table 2, we report the performance of RL algorithms using multiple flexibility settings, i.e., with/without handicap and action choices = 2. The performance of algorithms with a handicapped version of the environment seems to be easier when compared to a non-handicapped version, depicting the choice of keeping the handicap feature to be useful. For settings without any handicap provided, we found the RPPO algorithm to beat other RL algorithms by a significant margin. Fig. 4 shows the performance of algorithms over multiple episodes, depicting the convergence rate. We observe that RPPO convergence is faster at a higher score, and DQN seems unstable during initial episodes. We also plot performance curves for all the scenarios in App. D. As our RL framework combines language embeddings with RL algorithms, we also highlight the effect of different language model embeddings. We choose RPPO for reporting performance with different language models, as in extensive experimentation, we found RPPO to perform better than other RL algorithms on multiple environment settings. Fig. 5 reports the RPPO performance with different embeddings. We consider various types of SBERT-based embeddings (https://www.sbert.net/docs/pretrained_models.html). To judge the effect of contextual embeddings, we also report the RPPO performance with GloVe embeddings [Pennington *et al.*, 2014]. RPPO with GloVe embeddings (non-contextualized word representations) performs poorly, depicting the importance of the context which is captured by contextualized LMs (more results on LMs in App. D.)

Generalization across Scenarios: In `ScriptWorld` since all the scenarios belong to real-life daily activities, an interesting experiment is to test the generalization capability of an algorithm trained on a specific scenario. We chose two similar (in terms of commonsense knowledge required to solve) scenarios, `Going on a Train` and `Riding on a Bus`, for this experiment. Table 3 shows the evaluation

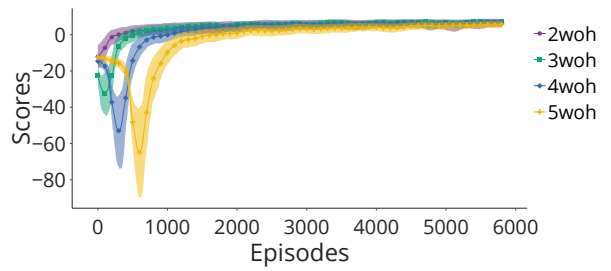

Figure 6: The figure shows the performance of RPPO algorithm on scenario `Repairing a Flat Bicycle Tire` (without handicap) on multiple choice settings, 2, 3, 4, 5 respectively.

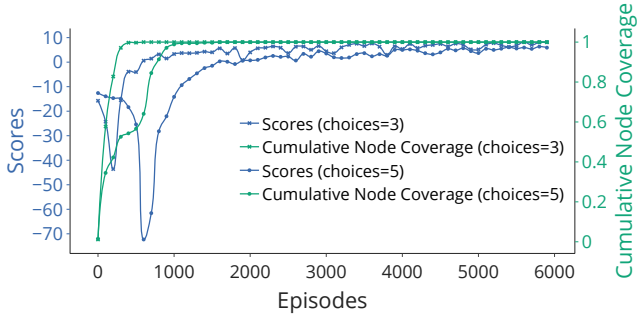

Figure 7: The figure shows the performance of RPPO algorithm on scenario `Repairing a Flat Bicycle Tire` (without handicap) on choices, 3 and, 5 with respective node coverages across learning. The increasing coverage slope (green) and the performance dip (blue) coincide in both settings highlighting the role of graph coverage in algorithm's learning.

performance of RPPO on all scenarios trained on one scenario. We observe that the RPPO algorithm generalizes more across similar scenarios e.g., between `Train` and `Bus` (more details in App. D). Results obtained in this experiment also open up new research directions like test-time domain adaptation and continual learning.

Performance on different choice settings: To benchmark the flexibility feature of choosing the number of actions in the environment setting, we also report the results for RPPO on various numbers of actions. Fig. 6 shows the training curves for settings with choices = 2, 3, 4, 5, highlighting the increasing difficulty level as the number of choices in the environment increases. We observe an interesting trend, the occurrence of a performance dip in all the scenarios for different episode numbers. Notice the performance dip in Fig. 6 for all the runs with varying numbers of choices. As can be observed, the episode for performance dip increases with the increasing number of choices in the environment. We study this behavior of RL algorithms in detail by analyzing the trajectory followed by the RL algorithms. Fig. 7 shows the percentage coverage of scenario graph nodes along with rewards. The point for a maximum dip (after which the algorithm starts improving the score) directly coincides with the increasing percentage of node coverage; we speculate that the algorithm begins developing a mapping for each node after the entire graph exploration and works on improving the node representation in the later episodes. Though the graph coverage percentage is higher, it still remains a difficult task to optimize for correct choice as the number of paths in the graph is huge, and the choices generated for each node are random, making each scenario node different at different time steps.

Training Scenario	Performance on other Scenarios								
	`Airplane`	`Bath`	`Bicycle`	**`Bus`**	`Cake`	`grocery`	`Haircut`	`Library`	**`Train`**	`Tree`
`Bus`	-24.78	-15.07	-5.02	**9.97**	-23.39	-26.51	-20.85	-16.29	**2.14**	-21.78
`Train`	-11.31	-13.22	-9.52	**5.44**	-4.42	0.22	-10.97	-6.79	**9.56**	-0.59

Table 3: The table shows performance on RPPO algorithm trained one scenario and evaluated on all scenarios. RPPO trained on `Bus` performs better on `Train` and vice versa (highlighted in red), depicting the generalization across scenarios.

6 Discussion and Future Directions

`ScriptWorld` provides a suitable benchmark to test different settings as it provides flexibility to adjust the game's complexity. The environment has certain limitations. For example, currently, the environment provides actions available at any state in the form of choices and does not allow the agent to generate actions in free-form text. This limitation is also there in the current parser based text-games that restrict the vocabulary size and sentence constructions that an agent can use for interaction. Parsing and understanding free-form text is a non-trivial task for the current state-of-the-art NLP technologies. In the future, we plan to develop a parser-based version (allowing free-form text) of the game, making use of LLMs. `ScriptWorld` 's current version only has 10 scenarios. This is mainly due to limitations from the DeScript corpus. In future work, we will try to address this by including more daily scenarios. Experiments show that agents struggle in no handicap setting since they do not have any prior knowledge about the real world. It would be interesting to incorporate external knowledge into agents in the future and explore the possibility of including human feedback for learning a new scenario. Alternatively, another idea to explore would be to allow agents to gather information about a task from the internet via search or by probing large language models. Including multiple diverse scenarios in the proposed environment can facilitate the validation of generalization and language understanding capabilities in fields like continual learning, where a single algorithm learns various tasks without catastrophic forgetting [Nguyen *et al.*, 2019].

7 Conclusion

In this paper, we present a novel approach to building a text-based game environment (`ScriptWorld`) involving different daily scenarios. This is a step towards training RL agents to develop NLU capabilities and commonsense knowledge about the real world. We perform an extensive set of experiments. Our experiments and analysis not only explore the environment in RL setting but also open up new ways in which the environment is helpful for the research community.

References

[Adhikari *et al.*, 2020] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikulas Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L. Hamilton. Learning Dynamic Belief Graphs to Generalize on Text-Based Games. In *NeurIPS*, 2020.

[Adolphs and Hofmann, 2020] Leonard Adolphs and Thomas Hofmann. LeDeepChef: Deep Reinforcement Learning Agent for Families of Text-Based Games. In *AAAI*, 2020.

[Ammanabrolu and Hausknecht, 2020] Prithviraj Ammanabrolu and Matthew Hausknecht. Graph Constrained Reinforcement Learning for Natural Language Action Spaces. In *International Conference on Learning Representations*, 2020.

[Ammanabrolu and Riedl, 2019] Prithviraj Ammanabrolu and Mark Riedl. Transfer in Deep Reinforcement Learning Using Knowledge Graphs. In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, 2019.

[Chatzou *et al.*, 2016] Maria Chatzou, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb, and Cedric Notredame. Multiple Sequence Alignment Modeling: Methods and Applications. *Briefings in Bioinformatics*, 2016.

[Chaudhury *et al.*, 2020] Subhajit Chaudhury, Daiki Kimura, Kartik Talamadupula, Michiaki Tatsubori, Asim Munawar, and Ryuki Tachibana. Bootstrapped Q-learning with Context Relevant Observation Pruning to Generalize in Text-based Games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[Côté *et al.*, 2018] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben A. Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A Learning Environment for Text-based Games. In *CGW@IJCAI*, 2018.

[Frermann *et al.*, 2014] Lea Frermann, Ivan Titov, and Manfred Pinkal. A Hierarchical Bayesian Model for Unsupervised Induction of Script Knowledge. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.

[Hausknecht *et al.*, 2020] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive Fiction Games: A Colossal Adventure. In *AAAI*, 2020.

[He *et al.*, 2016] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep Reinforcement Learning with a Natural Language Action Space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[Hill *et al.*, 2017] Felix Hill, Karl Moritz Hermann, Phil Blunsom, and Stephen Clark. Understanding Grounded Language Learning Agents. *arXiv preprint arXiv:1710.09867*, 2017.

[Jans *et al.*, 2012] Bram Jans, Steven Bethard, Ivan Vulić, and Marie Francine Moens. Skip N-grams and Ranking Functions for Predicting Script Events. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 2012.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 1998.

[Küttler *et al.*, 2020] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. In *NeurIPS*, 2020.

[Miikkulainen and Elman, 1993] Risto Miikkulainen and Jeffrey Elman. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT press, 1993.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, 2016.

[Modi and Titov, 2014] Ashutosh Modi and Ivan Titov. Inducing Neural Models of Script Knowledge. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, 2014.

[Modi *et al.*, 2016] Ashutosh Modi, Tatjana Anikina, Simon Ostermann, and Manfred Pinkal. InScript: Narrative texts annotated with script information. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016.

[Modi *et al.*, 2017] Ashutosh Modi, Ivan Titov, Vera Demberg, Asad Sayeed, and Manfred Pinkal. Modeling Semantic Expectation: Using Script Knowledge for Referent Prediction. *Transactions of the Association for Computational Linguistics*, 2017.

[Modi, 2016] Ashutosh Modi. Event Embeddings for Semantic Script Modeling. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, 2016.

[Modi, 2017] Ashutosh Modi. *Modeling Common Sense Knowledge via Scripts*. PhD thesis, Saarland University, 2017.

[Murugesan *et al.*, 2020] Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. Text-based RL Agents with Commonsense Knowledge: New Challenges,

Environments and Baselines. In *AAAI Conference on Artificial Intelligence*, 2020.

[Narasimhan *et al.*, 2015] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language Understanding for Text-based Games using Deep Reinforcement Learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[Nguyen *et al.*, 2019] Cuong V Nguyen, Alessandro Achille, Michael Lam, Tal Hassner, Vijay Mahadevan, and Stefano Soatto. Toward Understanding Catastrophic Forgetting in Continual Learning. *arXiv preprint arXiv:1908.01091*, abs/1908.01091, 2019.

[Ostermann *et al.*, 2018a] Simon Ostermann, Ashutosh Modi, Michael Roth, Stefan Thater, and Manfred Pinkal. MCScript: A Novel Dataset for Assessing Machine Comprehension Using Script Knowledge. In *LREC*, 2018.

[Ostermann *et al.*, 2018b] Simon Ostermann, Michael Roth, Ashutosh Modi, Stefan Thater, and Manfred Pinkal. SemEval-2018 Task 11: Machine Comprehension Using Commonsense Knowledge. In *Proceedings of the 12th International Workshop on Semantic Evaluation*, 2018.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[Pichotta and Mooney, 2016] Karl Pichotta and Raymond J. Mooney. Using Sentence-Level LSTM Language Models for Script Inference. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[Radford *et al.*, 2019] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Regneri *et al.*, 2010] Michaela Regneri, Alexander Koller, and Manfred Pinkal. Learning Script Knowledge with Web Experiments. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[Reimers and Gurevych, 2019] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[Rudinger *et al.*, 2015] Rachel Rudinger, Vera Demberg, Ashutosh Modi, Benjamin Van Durme, and Manfred Pinkal. Learning to predict script events from domain-specific text. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, 2015.

[Sakaguchi *et al.*, 2021] Keisuke Sakaguchi, Chandra Bhagavatula, Ronan Le Bras, Niket Tandon, Peter Clark, and Yejin Choi. proScript: Partially Ordered Scripts Generation. In *Findings of the Association for Computational Linguistics: EMNLP*, 2021.

[Sancheti and Rudinger, 2022] Abhilasha Sancheti and Rachel Rudinger. What do Large Language Models Learn about Scripts? In *Proceedings of the 11th Joint Conference on Lexical and Computational Semantics*, 2022.

[Schank and Abelson, 1975] Roger C. Schank and Robert P. Abelson. Scripts, Plans, and Knowledge. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, IJCAI, 1975.

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

[Singh *et al.*, 2022] Ishika Singh, Gargi Singh, and Ashutosh Modi. Pre-trained Language Models as Prior Knowledge for Playing Text-based Games. In *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2022.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[Wanzare *et al.*, 2016] Lilian D. A. Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. A Crowdsourced Database of Event Sequence Descriptions for the Acquisition of High-quality Script Knowledge. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016.

[Yao *et al.*, 2020] Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep CALM and Explore: Language Models for Action Generation in Text-based Games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[Yin and May, 2019] Xusen Yin and Jonathan May. Learn how to cook a new recipe in a new house: Using map familiarization, curriculum learning, and bandit feedback to learn families of text-based adventure games. *arXiv preprint arXiv:1908.04777*, 2019.

[Yuan *et al.*, 2019] Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Chris Pal, Yoshua Bengio, and Adam Trischler. Interactive Language Learning by Question Answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.