ScriptWorld: Text Based Environment For Learning Procedural Knowledge

Abhinav Joshi, Areeb Ahmad, Umang Pandey, Ashutosh Modi
Indian Institute of Technology Kanpur (IIT-K)
{ajoshi, ashuoshm}@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.

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 actions/segments 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, **ScriptWorld** consisting 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 freeform 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...
The game environment should be complex enough to allow for training RL agents. 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

**DeScript** provides a set of aligned ESDs $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_N$ for each scenario $\mathcal{S}$. Each ESD $\mathcal{E}_k$ consists of sequence of short event descriptions:

$$e_1, e_2, \ldots, e_n.$$  

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, the *Washing Dishes* scenario, events “puts dishes in sink” $(e_1^{Wash})$ in $\mathcal{E}_1^{Wash}$ and “take dirty dishes to sink” $(e_2^{Wash})$ 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**.

### Graph Formation

DeScript provides set of aligned ESDs $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_N$ for each scenario $\mathcal{S}$. Each ESD $\mathcal{E}_k$ consists of sequence of short event descriptions:

$$e_1, e_2, \ldots, e_n.$$  

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, the *Washing Dishes* scenario, events “puts dishes in sink” $(e_1^{Wash})$ in $\mathcal{E}_1^{Wash}$ and “take dirty dishes to sink” $(e_2^{Wash})$ 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**.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Nodes Deg. Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taking a Bath</td>
<td>525 3.7 1e+27</td>
</tr>
<tr>
<td>Baking a Cake</td>
<td>542 3.6 4.0e+26</td>
</tr>
<tr>
<td>Flying in an Airplane</td>
<td>528 3.6 2.6e+30</td>
</tr>
<tr>
<td>Going Grocery Shopping</td>
<td>544 3.7 2.3e+26</td>
</tr>
<tr>
<td>Going on a Train</td>
<td>427 3.7 4.1e+21</td>
</tr>
<tr>
<td>Planting a Tree</td>
<td>373 3.7 1.6e+16</td>
</tr>
<tr>
<td>Riding on a Bus</td>
<td>376 3.8 1.0e+17</td>
</tr>
<tr>
<td>Repairing Flat Bicycle Tire</td>
<td>402 3.4 8.4e+18</td>
</tr>
<tr>
<td>Borrowing Book from Library</td>
<td>397 3.7 8.1e+19</td>
</tr>
<tr>
<td>Getting a Haircut</td>
<td>528 3.7 4.0e+28</td>
</tr>
</tbody>
</table>

Table 1: The table compares graphs of different scenarios present in DeScript. Deg. represents the average degree for the nodes in the scenario graph.
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 = \( T = \sum_{p_k=0}^{N} n_1(p_k) \), 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 \( n_1(p_k) \) 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 hops for wrong actions from the table, the number of paths in each of the scenar-
Consequently, observations change. These parameters introduce as a penalty. For the start node, since backward hop is not action and may not follow the same path again, which acts hop distance of 1. Due to the presence of parallel paths in the experiments, agents played with the environment with a back-hop displacement whenever a wrong action is selected. When an agent hops as another game setting that decides how many hops to plan. It backward in the temporal domain, and this back-hop plan. It is defined by
\[ s_t \rightarrow a_t \rightarrow s_{t+1} \rightarrow \ldots \rightarrow s_T, \]

Going on a Train scenario "compact graph" created for the scenario Going on a Train.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>DQN</th>
<th>A2C</th>
<th>PPO</th>
<th>RPPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shopping</td>
<td>9.60 ± 0.62</td>
<td>-7.28 ± 11.15</td>
<td>9.90 ± 0.30</td>
<td>-9.81 ± 14.71</td>
</tr>
<tr>
<td>Bus</td>
<td>8.99 ± 0.79</td>
<td>-1.47 ± 11.16</td>
<td>9.89 ± 0.34</td>
<td>-7.37 ± 17.09</td>
</tr>
<tr>
<td>Train</td>
<td>9.21 ± 2.07</td>
<td>-3.10 ± 11.16</td>
<td>9.89 ± 0.31</td>
<td>-8.13 ± 14.99</td>
</tr>
<tr>
<td>Library</td>
<td>9.51 ± 0.68</td>
<td>-1.94 ± 9.87</td>
<td>9.88 ± 0.32</td>
<td>-3.03 ± 9.84</td>
</tr>
<tr>
<td>Haircut</td>
<td>9.86 ± 0.35</td>
<td>-9.30 ± 12.93</td>
<td>9.89 ± 0.34</td>
<td>-5.87 ± 12.28</td>
</tr>
<tr>
<td>Cake</td>
<td>9.32 ± 0.84</td>
<td>-4.13 ± 12.22</td>
<td>9.48 ± 0.92</td>
<td>-7.58 ± 13.18</td>
</tr>
<tr>
<td>Bicycle</td>
<td>9.50 ± 0.75</td>
<td>0.07 ± 7.89</td>
<td>9.95 ± 0.22</td>
<td>-3.49 ± 12.39</td>
</tr>
<tr>
<td>Tree</td>
<td>9.94 ± 0.24</td>
<td>-0.15 ± 7.83</td>
<td>9.86 ± 0.44</td>
<td>-3.54 ± 12.56</td>
</tr>
<tr>
<td>Airplane</td>
<td>9.68 ± 0.75</td>
<td>-4.21 ± 12.39</td>
<td>9.86 ± 0.35</td>
<td>-8.66 ± 12.66</td>
</tr>
<tr>
<td>Bath</td>
<td>9.68 ± 0.61</td>
<td>-4.49 ± 13.23</td>
<td>9.75 ± 0.57</td>
<td>-10.02 ± 15.95</td>
</tr>
</tbody>
</table>

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.

**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., TextWorld, 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 \( \Omega \) is the set of all actions (choices), \( \gamma \) is the set of observations, i.e., description of various actions, \( R \) is the reward obtained and \( \gamma \) is the discount parameter. The goal of an agent is to learn a policy \( \pi(a | s) \), i.e., a mapping from a set of observations to actions leading to an optimal choice in a particular
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 $\pi$. Recent developments in RL have proposed an approximation of $\pi(a | s)$/q-value via a parameterized model that takes state (features) and actions (features) as input and produces the $\pi(a | s)$/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.,

$$c \in \{c_1, \ldots, c_n\}; \ h_i = \text{LM}(c_i)$$

$$O = h_1 \oplus h_2 \oplus \ldots \oplus h_n$$

Subsequently, the RL framework generates $\pi(a | s)/Q$ 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...
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.

### 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


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


