Reading to Learn

Victor Yuan Zhong

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2023

Reading Committee:
Luke Zettlemoyer, Chair
Dieter Fox
Hannaneh Hajishirzi

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Reading to Learn

Victor Yuan Zhong

Chair of the Supervisory Committee:

Luke Zettlemoyer
Computer Science and Engineering

Traditional machine learning systems are trained on vast quantities of annotated data or experience. These systems often do not generalize to new, related problems that emerge after training, such as conversing about new topics or interacting with new environments. This thesis introduces Reading to Learn, a new class of algorithms that improve generalization by learning to read language specifications, without requiring any actual experience or labeled examples. This includes, for example, reading FAQ documents to learn to answer questions about new topics and reading manuals to learn to play new games. This thesis discusses new algorithms and data for Reading to Learn applied to a broad range of tasks, including policy learning in grounded environments and data synthesis for code generation, while also highlighting open challenges for this line of work. Ultimately, the goal of Reading to Learn is to democratize AI by making it accessible for low-resource problems where the practitioner cannot obtain annotated data at scale, but can instead write language specifications that models read to generalize.

Acknowledgements

My PhD would not have been possible without Luke Zettlemoyer. I thank Luke for being a tremendous advisor, a sincere advocate, and a trusted counselor. As academics, we are privileged to have the freedom to investigate questions that further human knowledge. From Luke, I learned to appreciate and respect the process of scientific research. Luke is not only a role-model scientist, he is a role-model person. Throughout the years I have witnessed first hand his kindness, generosity, and graciousness towards his students, his colleagues, and strangers he had just met. I am fortunate to have Luke as a role-model, a mentor, and a dear friend as I continue to the next phase of my career.

I am honoured to be a part of the University of Washington, the UW NLP group, and ZLab. I especially thank Akari Asai, Barghavi Paranjape, Eunsol Choi, Ivan Evtimov, Judit Acs, Gabor Szabo, Jesse Thomason, Julian Michael, Jungo Kasai, Mandar Joshi, Margaret Li, Mohit Shridhar, Ofir Press, Omer Levy, Sally Dong, Sewon Min, Srini Iyer, Suchin Gururangan, Tao Yu, Terra Blevins, Tim Dettmers, Weijia Shi, and Yizhong Wang for their friendship. Thank you especially to Akari, Ivan, Judit+Gabor, Mohit, Sally, Sewon, Tim, and Yizhong for the dinner parties, summer hikes, road trips, and COVID memories. I am also grateful to the excellent staff at the UW CSE department, especially Chiemi Yamaoka, Elise deGoede Dorough, Elle Brown, and Joe Eckert for guiding this clueless student towards graduation. The Allen School is a model department, much thanks to the leadership by Hank Levy and now Magda Bałazińska. I also thank the many student leaders that kept social events running, even through the pandemic. I could not have chosen a better place to complete my PhD.

Much of my thesis work was completed with the help with external collaborators. I thank Sebastian Riedel for allowing me into the FAIR London family, where I had the great fortune of learning from Tim Rocktäschel and Ed Grefenstette. My early work with Tim and Ed kickstarted my imagination of what roles

language can play in the context of learning, which resulted in much of the work that make up this thesis. I am also fortunate to have collaborated with Karthik Narasimhan and Yoav Artzi, both of whom I am grateful to now consider my mentors. I am also grateful to have worked with Austin Wang and Machel Reid, both creative and talented young researchers. My work was generously supported by FAIR, where I was fortunate to work with Mike Lewis, Scott Yih, and Sida Wang. My work was also generously supported by the Apple AI/ML Scholar Fellowship.

My path to the University of Washington would not have been possible without Chris Manning and Richard Socher. Coming out of my undergraduate studies, I was a hardware engineer by training. I thank Chris and Richard for taking a chance on me as a NLP researcher. I thank Caiming Xiong for his guidance during the early years of Salesforce Research, as well as his continued support and friendship. I also thank Bryan McCann, Audrey Cook, Melvin Gruesbeck, James Bradbury, Alex Trott, and Stephen Merity for the wonderful formative years of MetaMind and Salesforce Research. There are many people at Stanford and the University of Toronto to whom I am grateful for helping in my research career. Apart from those already mentioned, I thank Danqi Chen, Gabor Angeli, and Yuhao Zhang at Stanford, and Ana Klimovic, Jimmy Ba, Jonathan Rose, Shane Gu, and Zeb Tate at the University of Toronto.

Most importantly, I am immensely grateful to my parents Yiping Wang and Zhangdui Zhong, without whom my achievements would not have been possible. It has been a long path to this PhD. I thank my parents for their unconditional love, support, and patience. I left China when I was ten years old to a foreign land, but through all these years I have never felt out-of-place, never felt disadvantaged, and never felt alone. This is because of the sacrifice my parents made so that I can pursue not only the education I wanted, but the life I wanted. They taught me patience, hard work, and most importantly resilience. They have always believed in me, sometimes even more than I believed in myself — for this I thank them dearly.

Lastly, I thank Jiayi Li for her love and support. She is caring and wise beyond description. I am inspired by her kindness, her sincerity, and her perseverance. These past few years have been some of the happiest and most rewarding years of my life thanks to Jiayi.

DEDICATION

To my family, for supporting me always.

Contents

Chapter 1

Introduction

How can we teach machines to read as humans do? Instead of creating problem-specific training datasets and environments, we should simply be able to describe a problem or strategy in natural language and have the model learn by reading. This thesis decribes the **Reading to Learn** (R2L) framework, a new class of algorithms that improve generalization by learning to read language specifications, without requiring any actual experience or labeled examples.

1.1 Learning to Read models

1.1.1 Inference

Machine learning models are typically trained on vast quantities of annotated data or experience, but still do not generalize to new problems. Consider how we typically use a machine learning model in the context of natural language processing. Given a trained model F, the practitioner performs inference with the model according to Figure 1.1. In this inference procedure, we give the model F input text x, the model processes this input text and produces some output text y. In the context of question anwering, x may be a natural language question such as "how do I cook boeuf bourguinon?", which results in the model generating the recipe for the requested dish. In the context of code generation, or semantic parsing, x may be a natural language utterance such as "list products that the University of Washington purchased in 2022", which results in the model generating the corresponding SQL query over some database. In the rest of this thesis,

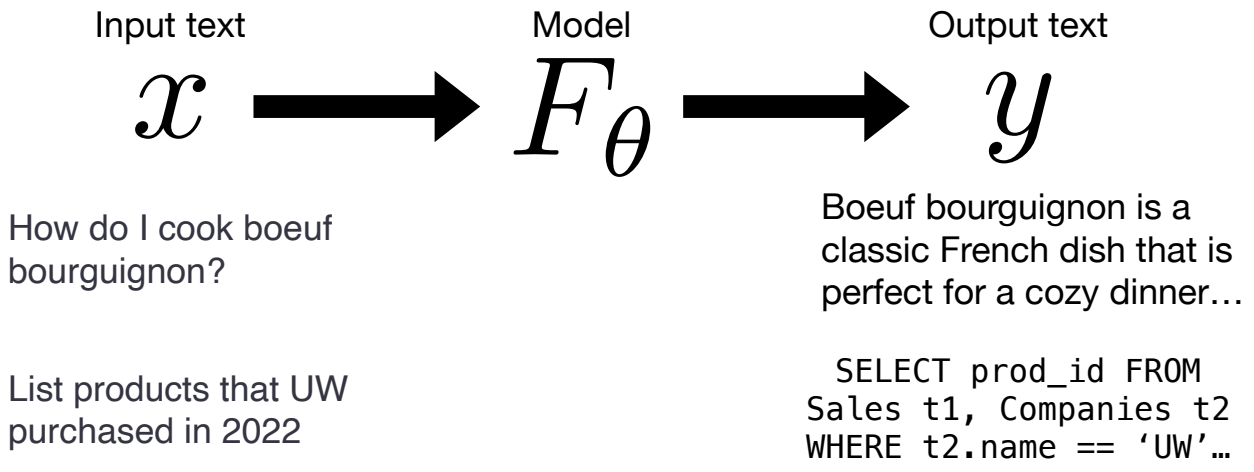


Figure 1.1: Inference with Learning to Read models.

we will refer to this framework of training a model to produce output text given input text as **Learning to Read** (L2R).

Learning to Read is a powerful framework in that it is applicable to many different problems. The SuperGLUE benchmark, a diverse selection of 10 language understanding datasets ranging from common-sense question-answering to textual entailment, is a prime example that illustrates the success of Learning to Read. These datasets, some of which were considered to be central challenges to language understanding as recently as ten years ago (i.e. 2014), have been largely been solved by the latest Learning to Read models. Perhaps even more convincingly, the models that solve these datasets are converging to similar large language model architectures (LLMs).

1.1.2 Training

Given the success of Reading to Learn models, how do we train them? Typically, we start by collecting a dataset of input-output pairs for the problem we would like to solve. Using this dataset, we optimize the parameters of the model: given the input, we want to find the parameters θ^* of the model F that maximizes the likelihood of the output. If we assume independence between each input-output pair in the dataset, then this procedure is equivalent to maximizing the product of the likelihood of each example in the dataset. This training procedure is shown in Eq (1.2). Once we find the best parameters θ of the model F, we can take

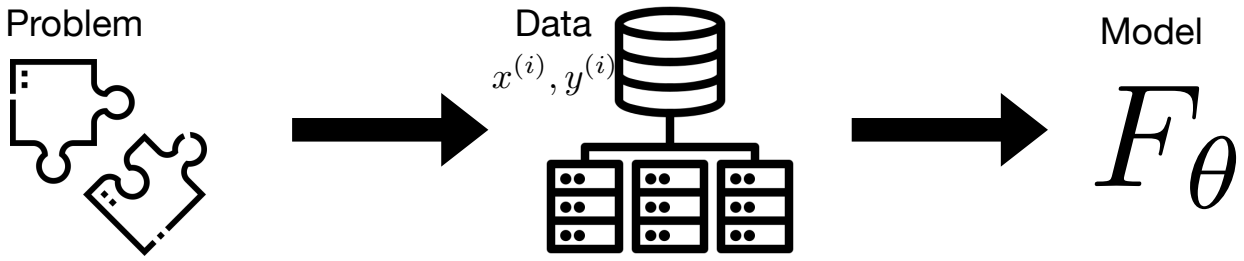


Figure 1.2: Training Learning to Read models.

this model and apply it to predict outputs for new inputs.

$$\theta^* = \text{argmax}_\theta F(\text{output}|\text{input};\theta) \tag{1.1}$$

$$= \text{argmax}_\theta \prod_{i=1}^{n} F(y^{(i)}|x^{(i)};\theta) \tag{1.2}$$

1.1.3 Limitations

Eq (1.2) illustrates the key limitation of Learning to Read models — in order to estimate a good parameters θ^* for the model, the practitioner typically needs to train on a vast quantity of input-output pairs. In other words, n tends to be very large: tasks such as sentiment analysis, textual entailment, and question-answering require hundreds of thousands of examples to train on. When the model is given a new problem, even if the new problem is only slightly different than the training problems, the practitioner typically needs to collect new labeled input-output pairs to train a new model. Consequently, while Learning to Read models are useful for many problems, they are often expensive to apply in practice due to this need for labeled datasets.

1.2 Reading to Learn

How can we avoid the need for large labeled datasets when the problem changes? In Reading to Learn, the model generalizes to new problems by reading problem descriptions. Given a problem, in the Reading to Learn framework, we find language specifications that describe the problem in the form of **manuals** that describe at a high level what the problem is and how the algorithm should behave. In practice, examples

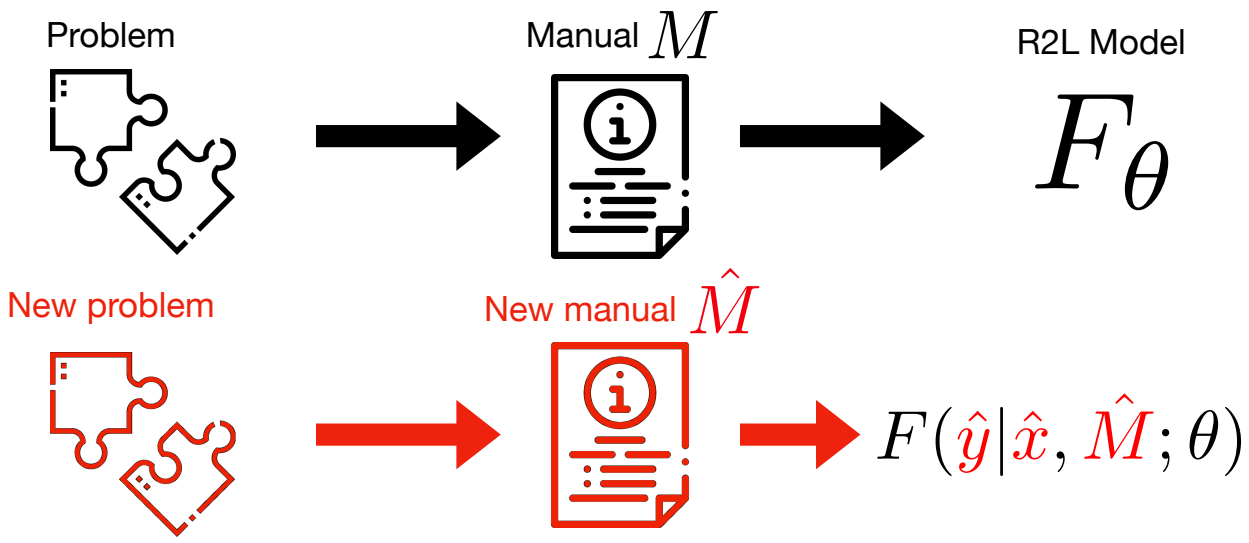
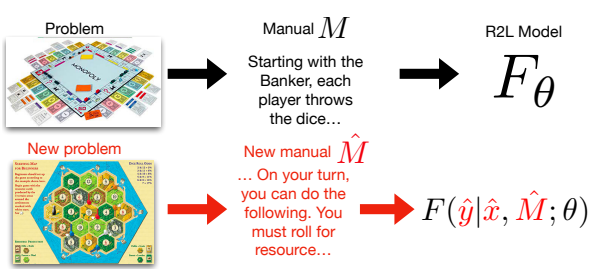


Figure 1.3: Generalization during inference with Reading to Learn models. The new problem, manual, input, and output are shown in red.

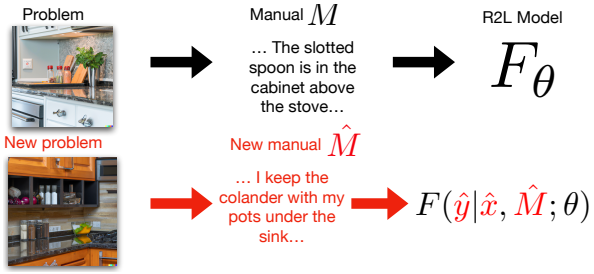
of these manuals include actual manuals (e.g. for tools, APIs), online wikis (e.g. for games), and human written guides (e.g. tutorials). As shown in Figure 1.3, the Reading to Learn model F reads the manual to learn about the problem and formulate a solution. Given a new problem, we find the corresponding manual that describes the new problem. Without additional training, the Reading to Learn model makes predictions for the new problem by reading its manual. Figure 1.4 shows examples of what Reading to Learn looks like in practice. In the left figure we have two different board games with different **environment dynamics**, that is, how actions affect the environment. Training on the first game Monopoly does not transfer to playing the second game Settlers of Catan. In Reading to Learn, the model reads manuals about the rules of Setters to figure out how to play it. In the right figure we have two kitchens with different layouts, different appliances, and potentially different ways of operating the appliances. Training in the first kitchen does not transfer to interacting with the second kitchen. In Reading to Learn, the model reads a text manual about what is in the kitchen and how to operate its appliances to figure out how to cook in the new kitchen.

1.2.1 Differences between Reading to Learn and Learning to Read

How does Reading to Learn differ from Learning to Read? Figure 1.5 shows differences between Reading to Learn and Learning to Read.

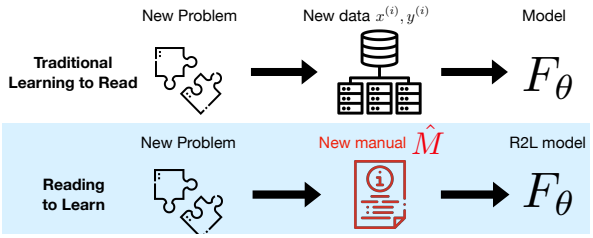


(a) Playing new board games by reading their rule manuals.

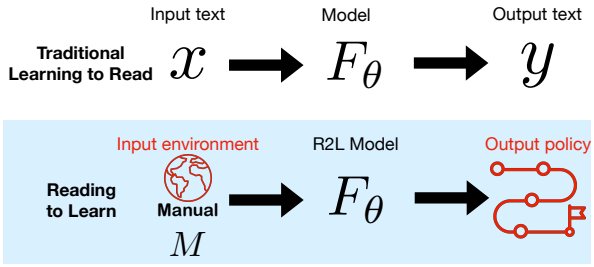


(b) Interacting in new kitchens by reading about their layout and appliances.

Figure 1.4: Examples of manuals that R2L models read to generalize.



(a) R2L generalizes without new training examples.



(b) R2L extends to grounded environments by reading environment manuals.

Figure 1.5: Differences between Reading to Learning and Learning to Read.

The first difference is in generalization . A Learning to Read model requires additional training data in order to solve to new problems. In contrast, when given a new problem, a Reading to Learn model requires only the corresponding new problem manual to solve the new problem. It is important to note that Reading to Learn additionally requires the collection, and possibly annotation, of problem manuals. However, we will show throughout this thesis that collecting these problem manuals is a much more cost-effective means to solve to new problems than collecting large labeled datasets.

The second difference is in application to grounded environments . Learning to read models have many applications, but they are fundamentally models of text: they process input text and produces output text. Reading to Learn models can also read text and generate text, so we can exploit the advances in new Large Language Models such as BERT and ChatGPT [Devlin et al., 2019; OpenAI, 2023]. But more than generating text, Reading to Learn is grounded in the world — we can read about manuals of an interactive environment to make decisions that affect the environment. For instance we can read about the layout of a kitchen to act in the kitchen according to instructions. In later results, we will see applications of Reading to Learn for policies that are grounded in simulated interactive kitchens, real-scene navigation, and real computer games.

Promise and challenges The promise of Reading to Learn is to make machine learning accessible for low-resource problems that are grounded in the real world. In these cases, annotated datasets of input-output pairs or step-by-step guidance on how to behave can be very expensive to collect. In this thesis, we propose teaching machines to read language manuals to solve problems, as humans do. What are some central challenges to building Reading to Learn models? We will highlight three challenges in particular.

1.2.2 Summary

This section presented an overview of recent machine machine learning techniques for natural language processing. In particular, we discussed the success of Learning to Read models that solve increasingly complex tasks by learning from large quantities of annotated input-output pairs. Next, we introduced an alternative learning framework in Reading to Learn, where a model learns to interpret problem manuals to solve problems. When faced with new problems, the Reading to Learn model interprets new problem

manuals to solve the new problems without needed additional annotated data.

In the remaining sections of this thesis, we will cover three challenges in Reading to Learn. These challenges are

1. How to test for generalization to new problems by reading? We will introduce tests for Reading to Learn via a reference game designed to evaluate generalizing to new problems by reading game manuals.

2. Once we have tests for Reading to Learn, how can we build general Reading to Learn models that work across different types of problems? We will extend the reference game setting to a benchmark of five Reading to Learn environments with distinct language grounding challenges.

3. How can we train Reading to Learn models efficiently, so that we can use them for grounded problems for which it is very expensive to collect a diverse set of problems and manuals? We will look at pretraining techniques that allow us to efficiently train Reading to Learn for grounded environments, using many fewer environments and manuals.

In the final section, we will conclude by discussion future works in Reading to Learn.

Chapter 2

Tests for generalization by reading

How might we test for generalization via reading? Consider the example in Figure 2.1 where we train a cooking robot that follows language instructions. Given this scenario, we may consider three types of generalization. First, does this robot generalize to new goals? For instance, can this robot follow new instructions in the same kitchen? Second, does this robot generalize to new variations of the problem? For instance, can this robot follow instructions if we re-arrange items in the kitchen? Third, does this robot generalize to new latent environment dynamics? For instance, if we have a new tool to drain liquid, or if we require a new operation to turn on the stove. In this section, we will formalize tests that evaluate these types of generalization. Figure 2.2 illustrates generalization to new latent environment dynamics. On the left, we train a model on the one kitchen. On the right, we take this trained model and have it follow instructions in the second kitchen with new latent environment dynamics. We will first examine generalization to new environment dynamics in a game setting, then formulate a more general benchmark for Reading to Learn.

2.1 Introduction

Reinforcement learning (RL) has been successful in a variety of areas such as continuous control [Lillicrap et al., 2015], dialogue systems [Li et al., 2016], and game-playing [Mnih et al., 2013]. However, RL adoption in real-world problems is limited due to poor sample efficiency and failure to generalise to environments even slightly different from those seen during training. We explore language-conditioned policy learning, where agents use machine reading to discover strategies required to solve a task, thereby leveraging language as a

Figure 2.1: Challenge 1: How can we test for generalization to new problems by reading?

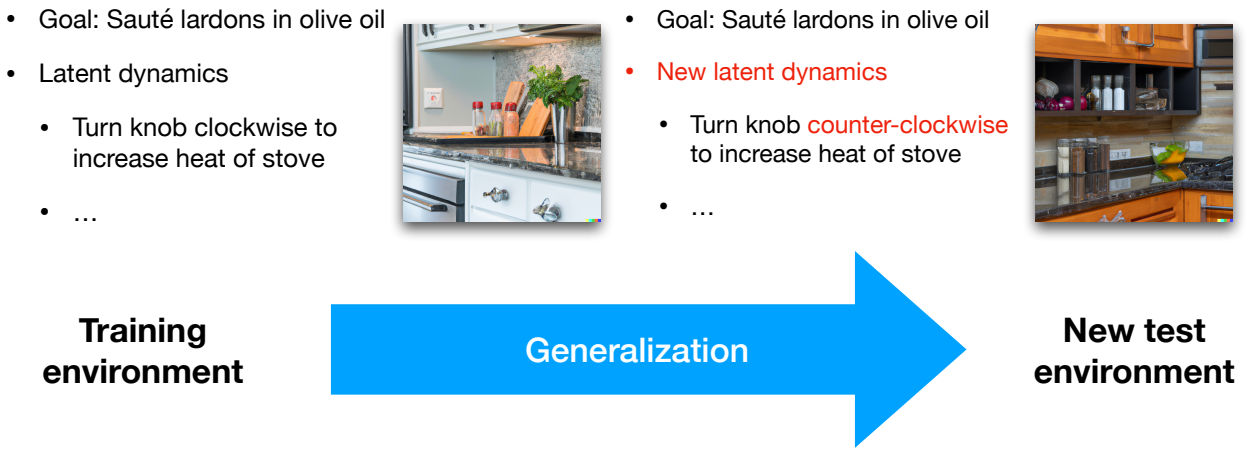


Figure 2.2: Generalization to new latent environment dynamics.

means to generalise to new environments.

Prior work on language grounding and language-based RL (see Luketina et al. [2019] for a recent survey) are limited to scenarios in which language specifies the goal for some fixed environment dynamics [Branavan et al., 2011; Hermann et al., 2017a; Bahdanau et al., 2019; Fried et al., 2018b; Co-Reyes et al., 2019], or the dynamics of the environment vary and are presented in language for some fixed goal [Branavan et al., 2012a]. In practice, changes to goals and to environment dynamics tend to occur simultaneously—given some goal, we need to find and interpret relevant information to understand how to achieve the goal. That is, the agent should account for variations in both by selectively reading, thereby generalising to environments with dynamics not seen during training.

Our contributions are two-fold. First, we propose a grounded policy learning problem that we call Reading to Fight Monsters (RTFM). In RTFM, the agent must jointly reason over a language goal, a document that specifies environment dynamics, and environment observations. In particular, it must identify relevant information in the document to shape its policy and accomplish the goal. To necessitate reading comprehension, we expose the agent to ever changing environment dynamics and corresponding language descriptions such that it cannot avoid reading by memorising any particular environment dynamics. We procedurally generate environment dynamics and natural language templated descriptions of dynamics and goals to produced a combinatorially large number of environment dynamics to train and evaluate RTFM.

Second, we propose txt2π to model the joint reasoning problem in RTFM. We show that txt2π generalises to goals and environment dynamics not seen during training, and outperforms previous language-conditioned models such as language-conditioned CNNs and FiLM [Perez et al., 2018; Bahdanau et al., 2019] both in terms of sample efficiency and final win-rate on RTFM. Through curriculum learning where we adapt txt2π trained on simpler tasks to more complex tasks, we obtain agents that generalise to tasks with natural language documents that require five hops of reasoning between the goal, document, and environment observations. Our qualitative analyses show that txt2π attends to parts of the document relevant to the goal and environment observations, and that the resulting agents exhibit complex behaviour such as retrieving correct items, engaging correct enemies after acquiring correct items, and avoiding incorrect enemies. Finally, we highlight the complexity of RTFM in scaling to longer documents, richer dynamics, and natural language variations. We show that significant improvement in language-grounded policy learning is

needed to solve these problems in the future.

2.2 Related Work

Language-conditioned policy learning. A growing body of research is learning policies that follow imperative instructions. The granularity of instructions vary from high-level instructions for application control [Branavan, 2012] and games [Hermann et al., 2017a; Bahdanau et al., 2019] to step-by-step navigation [Fried et al., 2018b]. In contrast to learning policies for imperative instructions, Branavan et al. [2011, 2012a]; Narasimhan et al. [2018b] infer a policy for a fixed goal using features extracted from high level strategy descriptions and general information about domain dynamics. Unlike prior work, we study the combination of imperative instructions and descriptions of dynamics. Furthermore, we require that the agent learn to filter out irrelevant information to focus on dynamics relevant to accomplishing the goal.

Language grounding. Language grounding refers to interpreting language in a non-linguistic context. Examples of such context include images [Barnard & Forsyth, 2001], games [Chen & Mooney, 2008; Wang et al., 2016], robot control [Kollar et al., 2010; Tellex et al., 2011], and navigation [Anderson et al., 2018b]. We study language grounding in interactive games similar to Branavan [2012]; Hermann et al. [2017a] or Co-Reyes et al. [2019], where executable semantics are not provided and the agent must learn through experience. Unlike prior work, we require grounding between an underspecified goal, a document of environment dynamics, and world observations. In addition, we focus on generalisation to not only new goal descriptions but new environments dynamics.

2.3 Reading to Fight Monsters

We consider a scenario where the agent must jointly reason over a language **goal**, relevant environment **dynamics** specified in a text document, and **environment observations**. In reading the document, the agent should identify relevant information key to solving the goal in the environment. A successful agent needs to perform this language grounding to generalise to new environments with dynamics not seen during training.

To study generalisation via reading, the environment dynamics must differ every episode such that the

Figure 2.3: RTFM requires jointly reasoning over the goal, a document describing environment dynamics, and environment observations. This figure shows key snapshots from a trained policy on one randomly sampled environment. Frame 1 shows the initial world. In 4, the agent approaches "fanatical sword", which beats the target "fire goblin". In 5, the agent acquires the sword. In 10, the agent evades the distractor "poison bat" while chasing the target. In 11, the agent engages the target and defeats it, thereby winning the episode. Sprites are used for visualisation — the agent observes cell content in text (shown in white). More examples are in Section 2.6.

agent cannot avoid reading by memorising a limited set of dynamics. Consequently, we procedurally generate a large number of unique environment dynamics (e.g. `effective(blessed items, poison monsters)`), along with language descriptions of environment dynamics (e.g. blessed items are effective against poison monsters) and goals (e.g. Defeat the order of the forest). We couple a large, customisable ontology inspired by rogue-like games such as NetHack or Diablo, with natural language templates to create a combinatorially rich set of environment dynamics to learn from and evaluate on.

In RTFM, the agent is given a document of environment dynamics, observations of the environment, and an underspecified goal instruction. Figure 2.3 illustrates an instance of the game. Concretely, we design a set of dynamics that consists of monsters (e.g. wolf, goblin), teams (e.g. Order of the Forest), element

types (e.g. fire, poison), item modifiers (e.g. fanatical, arcane), and items (e.g. sword, hammer). When the player is in the same cell with a monster or weapon, the player picks up the item or engages in combat with the monster. The player can possess one item at a time, and drops existing weapons if they pick up a new weapon. A monster moves towards the player with 60% probability, and otherwise moves randomly. The dynamics, the agent's inventory, and the underspecified goal are rendered as text. The game world is rendered as a matrix of text in which each cell describes the entity occupying the cell. We use human-written templates for stating which monsters belong to which team, which modifiers are effective against which element, and which team the agent should defeat (see Section 2.13 for details on collection and 2.12 for a list of entities in the game). In order to achieve the goal, the agent must cross-reference relevant information in the document and as well as in the observations.

During every episode, we subsample a set of groups, monsters, modifiers, and elements to use. We randomly generate group assignments of which monsters belong to which team and which modifier is effective against which element. A document that consists of randomly ordered statements corresponding to this group assignment is presented to the agent. We sample one element, one team, and a monster from that team (e.g. "fire goblin" from "Order of the forest") to be the target monster. Additionally, we sample one modifier that beats the element and an item to be the item that defeats the target monster (e.g. "fanatical sword"). Similarly, we sample an element, a team, and a monster from a different team to be the distractor monster (e.g. poison bat), as well as an item that defeats the distractor monster (e.g. arcane hammer).

In order to win the game (e.g. Figure 2.3), the agent must

1. identify the target team from the goal (e.g. Order of the Forest)
2. identify the monsters that belong to that team (e.g. goblin, jaguar, and ghost)
3. identify which monster is in the world (e.g. goblin), and its element (e.g. fire)
4. identify the modifiers that are effective against this element (e.g. fanatical, shimmering)
5. find which modifier is present (e.g. fanatical), and the item with the modifier (e.g. sword)
6. pick up the correct item (e.g. fanatical sword)
7. engage the correct monster in combat (e.g. fire goblin).

If the agent deviates from this trajectory (e.g. does not have correct item before engaging in combat, engages with distractor monster), it cannot defeat the target monster and therefore will lose the game. The agent receives a reward of +1 if it wins the game and -1 otherwise.

RTFM presents challenges not found in prior work in that it requires a large number of grounding steps in order to solve a task. In order to perform this grounding, the agent must jointly reason over a language goal and document of dynamics, as well as environment observations. In addition to the environment, the positions of the target and distractor within the document are randomised—the agent cannot memorise ordering patterns in order to solve the grounding problems, and must instead identify information relevant to the goal and environment at hand.

We split environments into train and eval sets. No assignments of monster-team-modifier-element are shared between train and eval to test whether the agent is able to generalise to new environments with dynamics not seen during training via reading. There are more than 2 million train or eval environments without considering the natural language templates, and 200 million otherwise. With random ordering of templates, the number of unique documents exceeds 15 billion.

2.4 Model

We propose the txt2π model, which builds representations that capture three-way interactions between the goal, document describing environment dynamics, and environment observations. We begin with definition of the Bidirectional Feature-wise Linear Modulation (FiLM^2) layer, which forms the core of our model.

2.4.1 Bidirectional Feature-wise Linear Modulation (FiLM^2) layer

Feature-wise linear modulation (FiLM), which modulates visual inputs using representations of textual instructions, is an effective method for image captioning [Perez et al., 2018] and instruction following [Bahdanau et al., 2019]. In RTFM, the agent must not only filter concepts in the visual domain using language but filter concepts in the text domain using visual observations. To support this, FiLM^2 builds codependent representations of text and visual inputs by further incorporating conditional representations of the text given visual observations. Figure 2.4 shows the FiLM^2 layer.

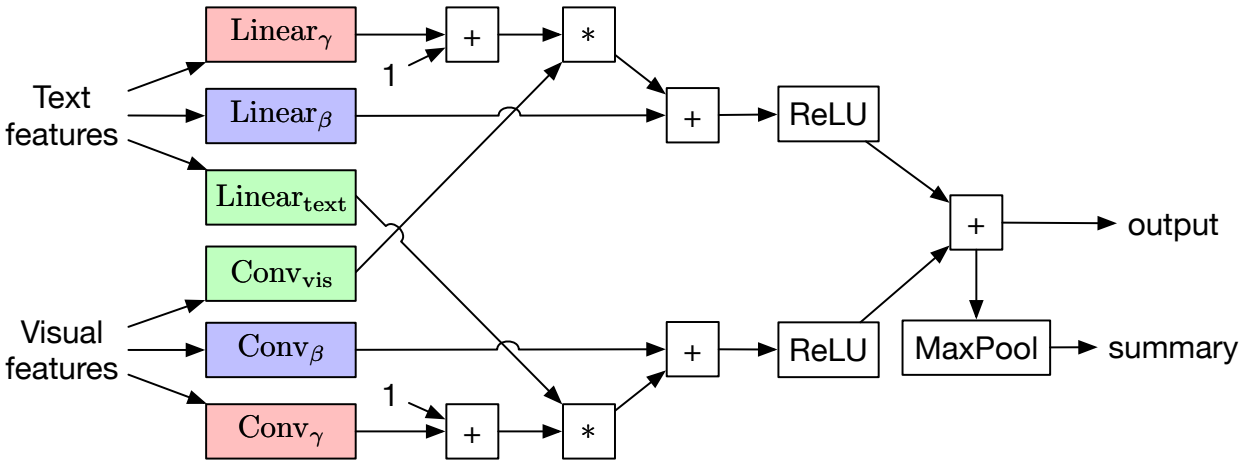


Figure 2.4: The FiLM^2 layer.

We use upper-case bold letters to denote tensors, lower-case bold letters for vectors, and non-bold letters for scalars. Exact dimensions of these variables are shown in Table 2.4 in Section 2.7.

Let $\boldsymbol{x}_{\text{text}}$ denote a fixed-length d_{text}-dimensional representation of the text and $\boldsymbol{X}_{\text{vis}}$ the representation of visual inputs with height H, width W, and d_{vis} channels. Let Conv denote a convolution layer. Let + and * symbols denote element-wise addition and multiplication operations that broadcast over spatial dimensions. We first modulate visual features using text features:

$$\boldsymbol{\gamma}_{\text{text}} = \boldsymbol{W}_\gamma \boldsymbol{x}_{\text{text}} + \boldsymbol{b}_\gamma \tag{2.1}$$

$$\boldsymbol{\beta}_{\text{text}} = \boldsymbol{W}_\beta \boldsymbol{x}_{\text{text}} + \boldsymbol{b}_\beta \tag{2.2}$$

$$\boldsymbol{V}_{\text{vis}} = \text{ReLU}((1 + \boldsymbol{\gamma}_{\text{text}}) * \text{Conv}_{\text{vis}}(\boldsymbol{X}_{\text{vis}}) + \boldsymbol{\beta}_{\text{text}}) \tag{2.3}$$

Unlike FiLM, we additionally modulate text features using visual features:

$$\boldsymbol{\Gamma}_{\text{vis}} = \text{Conv}_\gamma(\boldsymbol{X}_{\text{vis}}) \tag{2.4}$$

$$\boldsymbol{B}_{\text{vis}} = \text{Conv}_\beta(\boldsymbol{X}_{\text{vis}}) \tag{2.5}$$

$$\boldsymbol{V}_{\text{text}} = \text{ReLU}((1 + \boldsymbol{\Gamma}_{\text{vis}}) * (\boldsymbol{W}_{\text{text}} \boldsymbol{x}_{\text{text}} + \boldsymbol{b}_{\text{text}}) + \boldsymbol{B}_{\text{vis}}) \tag{2.6}$$

The output of the FiLM^2 layer consists of the sum of the modulated features $\boldsymbol{V}$, as well as a max-pooled sum-

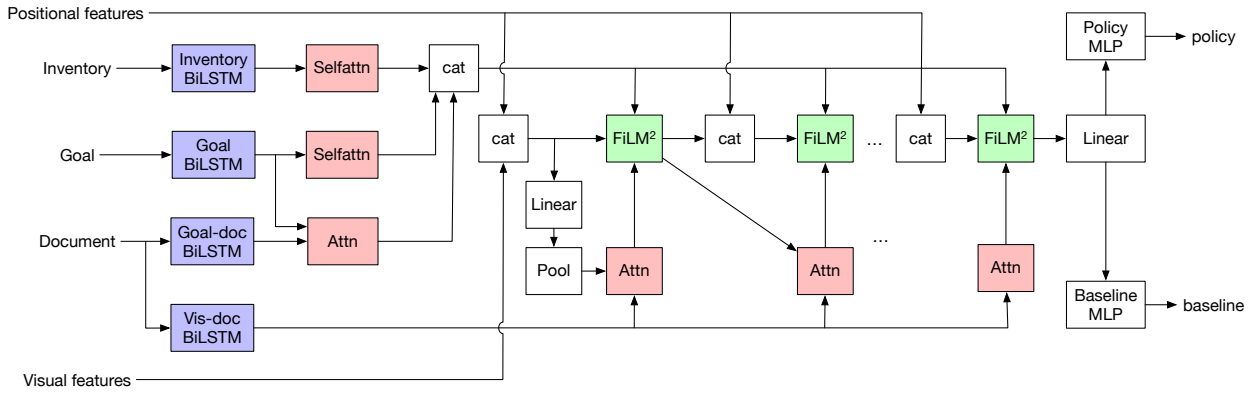


Figure 2.5: txt2π models interactions between the goal, document, and observations.

mary $\boldsymbol{s}$ over this sum across spatial dimensions.

$$\boldsymbol{V} = \boldsymbol{V}_{\text{vis}} + \boldsymbol{V}_{\text{text}} \qquad (2.7)$$

$\boldsymbol{s} = \text{Max}$

2.4.2 The txt2π model

We model interactions between observations from the environment, goal, and document using FiLM^2 layers. We first encode text inputs using bidirectional LSTMs, then compute summaries using self-attention and conditional summaries using attention. We concatenate text summaries into text features, which, along with visual features, are processed through consecutive FiLM^2 layers. In this case of a textual environment, we consider the grid of word embeddings as the visual features for FiLM^2. The final FiLM^2 output is further processed by MLPs to compute a policy distribution over actions and a baseline for advantage estimation. Figure 2.5 shows the txt2π model.

Let $\boldsymbol{E}_{\text{obs}}$ denote word embeddings corresponding to the observations from the environment, where $\boldsymbol{E}_{\text{obs}}[:, :, i, j]$ represents the embeddings corresponding to the l_{obs}-word string that describes the objects in location (i, j) in the grid-world. Let $\boldsymbol{E}_{\text{doc}}$, $\boldsymbol{E}_{\text{inv}}$, and $\boldsymbol{E}_{\text{goal}}$ respectively denote the embeddings corresponding to the l_{doc}-word document, the l_{inv}-word inventory, and the l_{goal}-word goal. We first compute a fixed-length summary $\boldsymbol{c}_{\text{goal}}$ of the the goal using a bidirectional LSTM [Hochreiter & Schmidhuber, 1997] followed by self-attention [Lee et al., 2017; Zhong et al., 2018].

$$\boldsymbol{H}_{\text{goal}} = \text{BiLSTM}_{\text{goal}}(\boldsymbol{E}_{\text{goal}}) \qquad (2.9)$$

$$a'_{\text{goal},i} = \boldsymbol{w}_{\text{goal}} \boldsymbol{h}^{\intercal}_{\text{goal},i} + b_{\text{goal}} \qquad (2.10)$$

$$\boldsymbol{a}_{\text{goal}} = \text{Softmax}(\boldsymbol{a}'_{\text{goal}}) \qquad (2.11)$$

$$\boldsymbol{c}_{\text{goal}} = \sum_{i=1}^{l_{\text{goal}}} a_{\text{goal},i} \boldsymbol{h}_{\text{goal},i} \qquad (2.12)$$

We abbreviate self-attention over the goal as $\boldsymbol{c}_{\text{goal}} = \text{selfattn}(\boldsymbol{H}_{\text{goal}})$. We similarly compute a summary of the inventory as $\boldsymbol{c}_{\text{inv}} = \text{selfattn}(\text{BiLSTM}_{\text{inv}}(\boldsymbol{E}_{\text{inv}}))$. Next, we represent the document encoding conditioned on the goal using dot-product attention [Luong et al., 2015].

$$\boldsymbol{H}_{\text{doc}} = \text{BiLSTM}_{\text{goal-doc}}(\boldsymbol{E}_{\text{doc}}) \tag{2.13}$$

$$a'_{\text{doc},i} = \boldsymbol{c}_{\text{goal}} \boldsymbol{h}_{\text{doc},i}^{\intercal} \tag{2.14}$$

$$\boldsymbol{a}_{\text{doc}} = \text{Softmax}(\boldsymbol{a}'_{\text{doc}}) \tag{2.15}$$

$$\boldsymbol{c}_{\text{doc}} = \sum_{i=1}^{l_{\text{doc}}} a_{\text{doc},i} \boldsymbol{h}_{\text{doc},i} \tag{2.16}$$

We abbreviate attention over the document encoding conditioned on the goal summary as $\boldsymbol{c}_{\text{doc}} = \text{attend}(\boldsymbol{H}_{\text{doc}}, \boldsymbol{c}_{\text{goal}})$. Next, we build the joint representation of the inputs using successive FiLM^2 layers. At each layer, the visual input to the FiLM^2 layer is the concatenation of the output of the previous layer with positional features. For each cell, the positional feature $\boldsymbol{X}_{\text{pos}}$ consists of the x and y distance from the cell to the agent's position respectively, normalized by the width and height of the grid-world. The text input is the concatenation of the goal summary, the inventory summary, the attention over the document given the goal, and the attention over the document given the previous visual summary. Let $[a; b]$ denote the feature-wise concatenation of a and b. For the ith layer, we have

$$R^{(i)} = [\boldsymbol{V}^{(i-1)}; \boldsymbol{X}_{\text{pos}}] \tag{2.17}$$

$$T^{(i)} = [\boldsymbol{c}_{\text{goal}}; \boldsymbol{c}_{\text{inv}}; \boldsymbol{c}_{\text{doc}}; \text{attend}(\text{BiLSTM}_{\text{vis-doc}}(\boldsymbol{E}_{\text{doc}}), \boldsymbol{s}^{(i-1)})] \tag{2.18}$$

$$\boldsymbol{V}^{(i)}, \boldsymbol{s}^{(i)} = \text{FiLM}^{2(\text{i})}(\text{R}^{(\text{i})}, \text{T}^{(\text{i})}) \tag{2.19}$$

$\text{BiLSTM}_{\text{vis-doc}}(\boldsymbol{E}_{\text{doc}})$ is another encoding of the document similar to $\boldsymbol{H}_{\text{goal}}$, produced using a separate LSTM, such that the document is encoded differently for attention with the visual features and with the goal. For $i = 0$, we concatenate the bag-of-words embeddings of the grid with positional features as the initial visual features $\boldsymbol{V}^{(0)} = [\sum_j \boldsymbol{E}_{\text{obs},j}; \boldsymbol{X}_{\text{pos}}]$. We max pool a linear transform of the initial visual features to compute the initial visual summary $\boldsymbol{s}^{(0)} = \text{MaxPool}(\boldsymbol{W}_{\text{ini}} \boldsymbol{V}^{(0)} + \boldsymbol{b}_{\text{ini}})$. Let $\boldsymbol{s}^{(\text{last})}$ denote visual

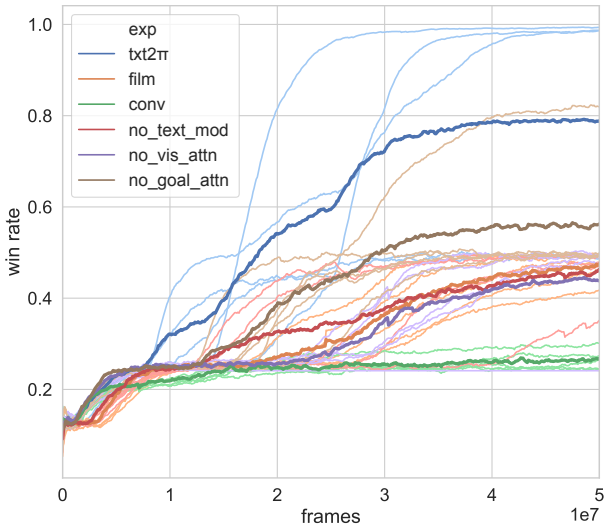


Figure 2.6: Ablation training curves on simplest variant of RTFM. Individual runs are in light colours. Average win rates are in bold, dark lines.

Model	Win rate		
	Train	Eval 6×6	Eval 10×10
conv	24 ± 0	25 ± 1	13 ± 1
FiLM	49 ± 1	49 ± 2	32 ± 3
`no_task_attn`	49 ± 2	49 ± 2	35 ± 6
`no_vis_attn`	49 ± 2	49 ± 1	40 ± 12
`no_text_mod`	49 ± 1	49 ± 2	35 ± 2
txt2π	84 ± 21	83 ± 21	66 ± 22

Table 2.1: Final win rate on simplest variant of RTFM. The models are trained on one set of dynamics (e.g. training set) and evaluated on another set of dynamics (e.g. evaluation set). "Train" and "Eval" show final win rates on training and eval environments.

summary of the last FiLM^2 layer. We compute the policy $\boldsymbol{y}_{\text{policy}}$ and baseline y_{baseline} as

$$\boldsymbol{o} = \text{ReLU}(\boldsymbol{W}_o \boldsymbol{s}^{(\text{last})} + \boldsymbol{b}_o) \tag{2.20}$$

$$\boldsymbol{y}_{\text{policy}} = \text{MLP}_{\text{policy}}(\boldsymbol{o}) \tag{2.21}$$

$$y_{\text{baseline}} = \text{MLP}_{\text{baseline}}(\boldsymbol{o}) \tag{2.22}$$

where $\text{MLP}_{\text{policy}}$ and $\text{MLP}_{\text{baseline}}$ are 2-layer multi-layer perceptrons with ReLU activation. We train using TorchBeast [Küttler et al., 2019b], an implementation of IMPALA [Espeholt et al., 2018c]. Please refer to Section 2.9 for details.

2.5 Experiments

We consider variants of RTFM by varying the size of the grid-world (6×6 vs 10×10), allowing many-to-one group assignments to make disambiguation more difficult (`group`), allowing dynamic, moving monsters that hunt down the player (`dyna`), and using natural language templated documents (`nl`). In the absence of many-to-one assignments, the agent does not need to perform steps 3 and 5 in section 2.3 as there is no need to disambiguate among many assignees, making it easier to identify relevant information.

We compare txt2π to the FiLM model by Bahdanau et al. [2019] and a language-conditioned residual CNN model. We train on one set of dynamics (e.g. group assignments of monsters and modifiers) and evaluated on a held-out set of dynamics. We also study three variants of txt2π. In `no_task_attn`, the document attention conditioned on the goal utterance ((2.16)) is removed and the goal instead represented through self-attention and concatenated with the rest of the text features. In `no_vis_attn`, we do not attend over the document given the visual output of the previous layer ((2.18)), and the document is instead represented through self-attention. In `no_text_mod`, text modulation using visual features ((2.6)) is removed. Please see Section 2.8 for model details on our model and baselines, and Section 2.9 for training details.

2.5.1 Comparison to baselines and ablations

We compare txt2π to baselines and ablated variants on a simplified variant of RTFM in which there are one-to-one group assignments (no `group`), stationary monsters (no `dyna`), and no natural language templated descriptions (no `nl`). Figure 2.6 shows that compared to baselines and ablated variants, txt2π is more sample efficient and converges to higher performance. Moreover, no ablated variant is able to solve the tasks—it is the combination of ablated features that enables txt2π to win consistently. Qualitatively, the ablated variants converge to locally optimum policies in which the agent often picks up a random item and then attacks the correct monster, resulting in a $\sim 50\%$ win rate. Table 2.1 shows that all models, with the exception of the CNN baseline, generalise to new evaluation environments with dynamics and world configurations not seen during training, with txt2π outperforming FiLM and the CNN model.

We find similar results for txt2π, its ablated variants, and baselines on a separate, language-based rock-paper-scissors task in which the agent needs to deduce cyclic dependencies (which type beats which other type) through reading in order to acquire the correct item and defeat a monster. We observe that the performance of reading models transfer from training environments to new environments with unseen types and unseen dependencies. Compared to ablated variants and baselines, txt2π is more sample efficient and achieves higher performance on both training and new environment dynamics. When transferring to new environments, txt2π remains more sample efficient than the other models. Details on these experiments are found in Section 2.10.

Transfer from	Transfer to							
	6×6	6×6 dyna	6×6 groups	6×6 nl	6×6 dyna groups	6×6 group nl	6×6 dyna nl	6×6 dyna group nl
random	$\mathbf{84 \pm 20}$	26 ± 7	25 ± 3	45 ± 6	23 ± 2	25 ± 3	23 ± 2	23 ± 2
+6×6		$\mathbf{85 \pm 9}$	82 ± 19	78 ± 24	64 ± 12	52 ± 13	53 ± 18	40 ± 8
+dyna					$\mathbf{77 \pm 10}$		65 ± 16	43 ± 4
+group								$\mathbf{65 \pm 17}$

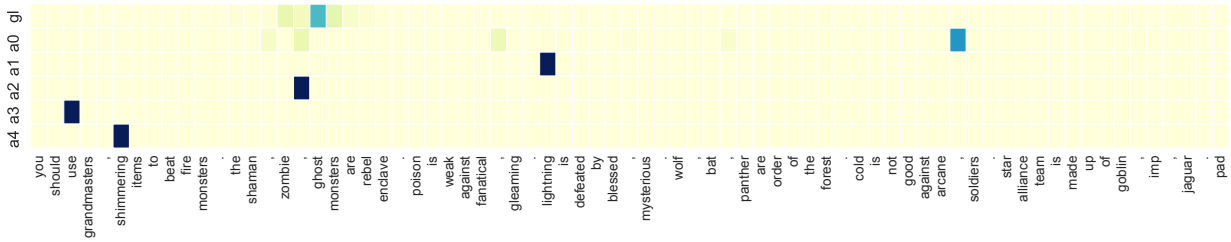
Table 2.2: Curriculum training results. We keep 5 randomly initialised models through the entire curriculum. A cell in row i and column j shows transfer from the best-performing setting in the previous stage (bold in row $i-1$) to the new setting in column j. Each cell shows final mean and standard deviation of win rate on the training environments. Each experiment trains for 50 million frames, except for the initial stage (first row, 100 million instead). For the last stage (row 4), we also transfer to a $10 \times 10 +$ dyna $+$ group $+$ nl variant and obtain $\mathbf{61 \pm 18}$ win rate.

Train env	Eval env	Win rate	
		Train	Eval
6×6	6×6	65 ± 17	55 ± 22
	10×10		55 ± 27
10×10	10×10	61 ± 18	43 ± 13

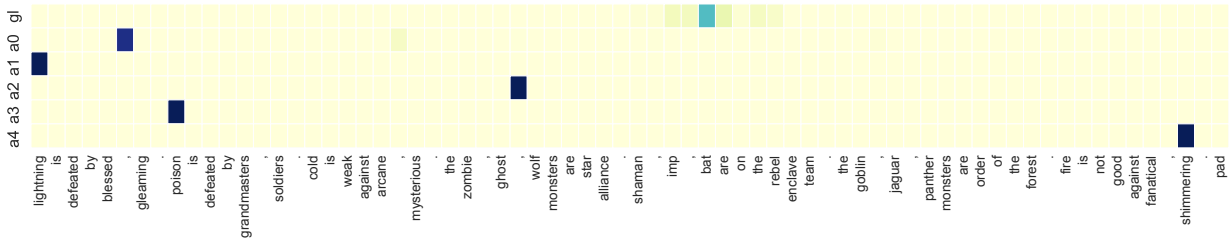
Table 2.3: Win rate when evaluating on new dynamics and world configurations for txt2π on the full RTFM problem.

2.5.2 Curriculum learning for complex environments

Due to the long sequence of co-references the agent must perform in order to solve the full RTFM (10×10 with moving monsters, many-to-one group assignments, and natural language templated documents) we design a curriculum to facilitate policy learning by starting with simpler variants of RTFM. We start with the simplest variant (no group, no dyna, no nl) and then add in an additional dimension of complexity. We repeatedly add more complexity until we obtain 10×10 worlds with moving monsters, many-to-one group assignments and natural language templated descriptions. The performance across the curriculum is shown in Table 2.2 (see Figure 2.15 in Section 2.11 for training curves of each stage). We see that curriculum learning is crucial to making progress on RTFM, and that initial policy training (first row of Table 2.2) with additional complexities in any of the dimensions result in significantly worse performance. We take each



(a) The entities present are shimmering morning star, mysterious spear, fire jaguar, and lightning ghost.



(b) The entities present are soldier's axe, shimmering axe, fire shaman, and poison wolf.

Figure 2.7: txt2π attention on the full RTFM. These include the document attention conditioned on the goal (top) as well as those conditioned on summaries produced by intermediate FiLM^2 layers. Weights are normalised across words (e.g. horizontally). Darker means higher attention weight.

of the 5 runs after training through the whole curriculum and evaluate them on dynamics not seen during training. Table 2.3 shows variants of the last stage of the curriculum in which the model was trained on 6×6 versions of the full RTFM and in which the model was trained on 10×10 versions of the full RTFM. We see that models trained on smaller worlds generalise to bigger worlds. Despite curriculum learning, however, performance of the final model trail that of human players, who can consistently solve RTFM. This highlights the difficulties of the RTFM problem and suggests that there is significant room for improvement in developing better language grounded policy learners.

Attention maps. Figure 2.7 shows attention conditioned on the goal and on observation summaries produced by intermediate FiLM^2 layers. Goal-conditioned attention consistently locates the clause that contains the team the agent is supposed to attack. Intermediate layer attentions focus on regions near modifiers and monsters, particularly those that are present in the observations. These results suggests that attention mechanisms in txt2π help identify relevant information in the document.

Analysis of trajectories and failure modes. We examine trajectories from well-performing policies (80% win rate) as well as poorly-performing policies (50% win rate) on the full RTFM. We find that well-performing policies exhibit a number of consistent behaviours such as identifying the correct item to pick up to fight the target monster, avoiding distractors, and engaging target monsters after acquiring the correct item. In contrast, the poorly-performing policies occasionally pick up the wrong item, causing the agent to lose when engaging with a monster. In addition, it occasionally gets stuck in evading monsters indefinitely, causing the agent to lose when the time runs out. Replays of both policies can be found in GIFs in the supplementary materials[1].

2.6 Playthrough examples

These figures shows key snapshots from a trained policy on randomly sampled environments.

2.7 Variable dimensions

Let $\boldsymbol{x}_{\text{text}} \in \mathbb{R}^{d_{\text{text}}}$ denote a fixed-length d_{text}-dimensional representation of the text and $\boldsymbol{X}_{\text{vis}} \in \mathbb{R}^{d_{\text{vis}} \times H \times W}$ denote the representation of visual inputs with

Variable	Symbol	Dimension
d_{text}-dim text representation	$\boldsymbol{x}_{\text{text}}$	d_{text}
d_{vis}-dim visual representation with height H, width W, d_{vis} channels	$\boldsymbol{X}_{\text{vis}}$	$d_{\text{vis}} \times H \times W$
Environment observations embeddings	$\boldsymbol{E}_{\text{obs}}$	$l_{\text{obs}} \times d_{\text{emb}} \times H \times W$
l_{obs}-word string that describes the objects in location (i, j) in the grid-world	$\boldsymbol{E}_{\text{obs}}[:, :, i, j]$	$l_{\text{obs}} \times d_{\text{emb}}$
l_{doc}-word document embeddings	$\boldsymbol{E}_{\text{doc}}$	$l_{\text{doc}} \times d_{\text{emb}}$
l_{inv}-word inventory embeddings	$\boldsymbol{E}_{\text{inv}}$	$l_{\text{inv}} \times d_{\text{emb}}$
l_{goal}-word goal embeddings	$\boldsymbol{E}_{\text{goal}}$	$l_{\text{goal}} \times d_{\text{emb}}$

Table 2.4: Variable dimensions

[1] Trajectories by txt2π on RTFM can be found at `https://gofile.io/?c=9k7ZLk`

Doc:
You should use arcane and mysterious items to beat lightning monsters. Panther, warg, and wolf are from the Order of the Forest. Blessed and Grandmaster's items beat poison monsters. Cold is not good against fanatical and shimmering weapons. The Star Alliance team is made up of beetle, jackal, and shaman. Imp, jaguar, and lynx make up the Rebel Enclave. Gleaming and Soldier's weapons beat fire monsters.
Goal:
Defeat the Star Alliance

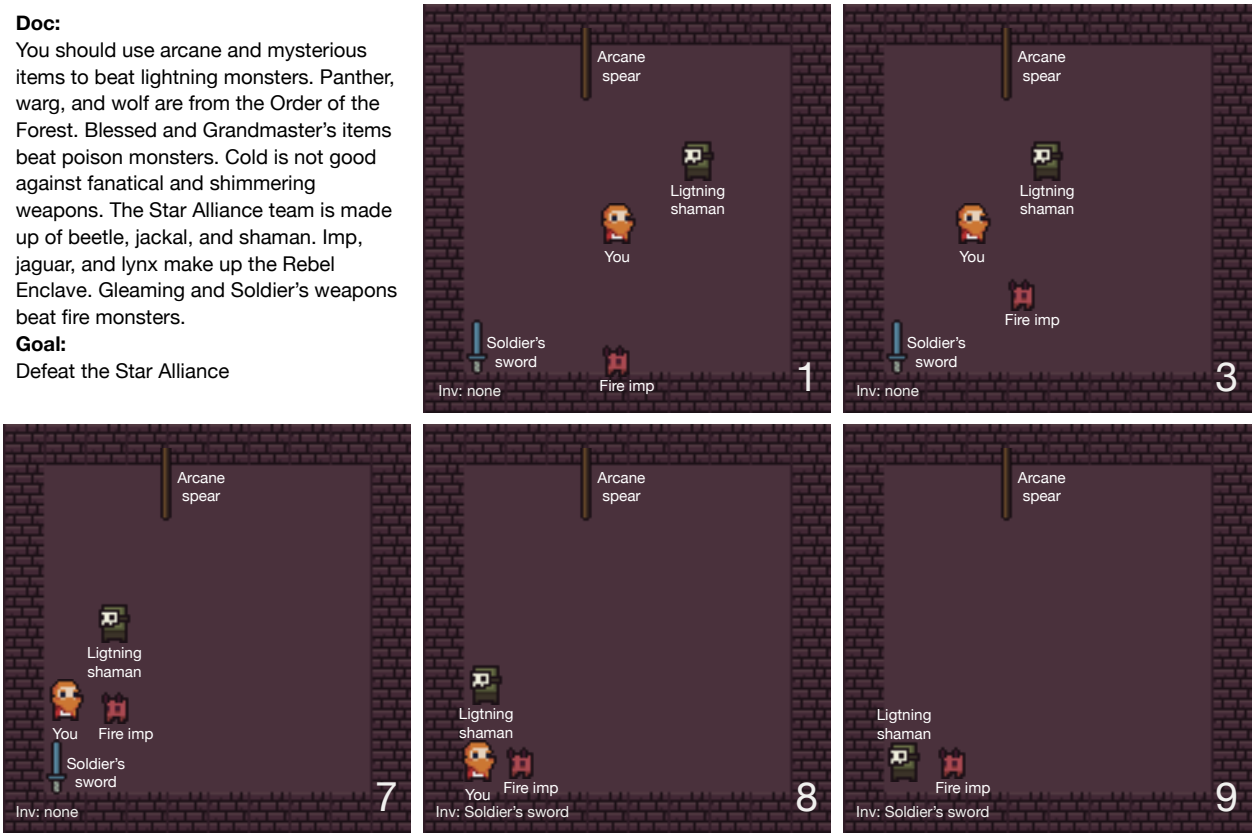


Figure 2.8: The initial world is shown in 1. In 4, the agent avoids the target "lightning shaman" because it does not yet have "arcane spear", which beats the target. In 7 and 8, the agent is cornered by monsters. In 9, the agent is forced to engage in combat and loses.

2.8 Model details

2.8.1 txt2π

Hyperparameters. The txt2π used in our experiments consists of 5 consecutive FiLM^2 layers, each with 3x3 convolutions and padding and stride sizes of 1. The txt2π layers have channels of 16, 32, 64, 64, and 64, with residual connections from the 3rd layer to the 5th layer. The Goal-doc LSTM (see Figure 2.5) shares weight with the Goal LSTM. The Inventory and Goal LSTMs have a hidden dimension of size 10, whereas the Vis-doc LSTM has a dimension of 100. We use a word embedding dimension of 30.

2.8.2 CNN with residual connections

Like txt2π, the CNN baseline consists of 5 layers of convolutions with channels of 16, 32, 64, 64, and 64. There are residual connections from the 3rd layer to the 5th layer. The input to each layer consists of the

Figure 2.9: The initial world is shown in 1. In 5 the agent evades the target "cold ghost" because it does not yet have "soldier's knife", which beats the target. In 11 and 13, the agent obtains "soldier's knife" while evading monsters. In 14, the agent defeats the target and wins.

output of the previous layer, concatenated with positional features.

The input to the network is the concatenation of the observations $\boldsymbol{V}^{(0)}$ and text representations. The text representations consist of self-attention over bidirectional LSTM-encoded goal, document, and inventory. These attention outputs are replicated over the dimensions of the grid and concatenated feature-wise with the observation embeddings in each cell. Figure 2.10 illustrates the CNN baseline.

2.8.3 FiLM baseline

The FiLM baseline encodes text in the same fashion as the CNN model. However, instead of using convolutional layers, each layer is a FiLM layer from Bahdanau et al. [2019]. Note that in our case, the language representation is a self-attention over the LSTM states instead of a concatenation of terminal LSTM states.

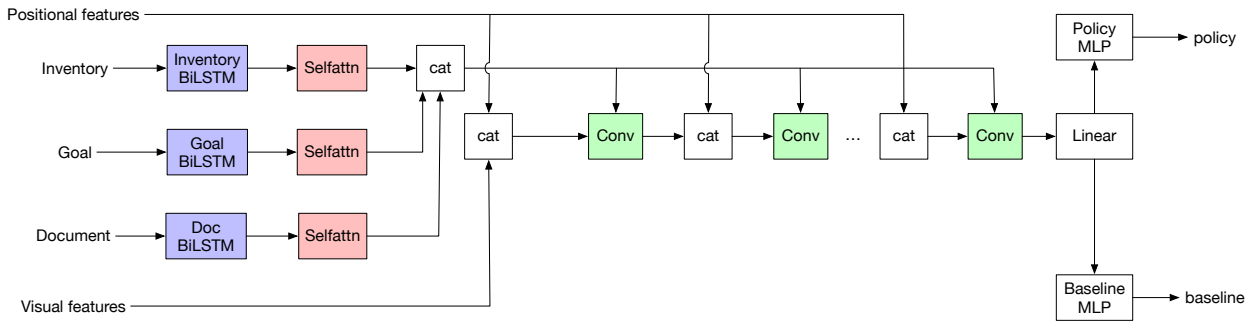


Figure 2.10: The convolutional network baseline. The FiLM baseline has the same structure, but with convolutional layers replaced by FiLM layers.

2.9 Training procedure

We train using an implementation of IMPALA [Espeholt et al., 2018c]. In particular, we use 20 actors and a batch size of 24. When unrolling actors, we use a maximum unroll length of 80 frames. Each episode lasts for a maximum of 1000 frames. We optimise using RMSProp [Tieleman & Hinton, 2012] with a learning rate of 0.005, which is annealed linearly for 100 million frames. We set $\alpha = 0.99$ and $\epsilon = 0.01$.

During training, we apply a small negative reward for each time step of -0.02 and a discount factor of 0.99 to facilitate convergence. We additionally include a entropy cost to encourage exploration. Let $\boldsymbol{y}_{\text{policy}}$ denote the policy. The entropy loss is calculated as

$$L_{\text{policy}} = -\sum_i \boldsymbol{y}_{\text{policy}_i} \log \boldsymbol{y}_{\text{policy}_i} \tag{2.23}$$

In addition to policy gradient, we add in the entropy loss with a weight of 0.005 and the baseline loss with a weight of 0.5. The baseline loss is computed as the root mean square of the advantages [Espeholt et al., 2018c].

When tuning models, we perform a grid search using the training environments to select hyperparameters for each model. We train 5 runs for each configuration in order to report the mean and standard deviation. When transferring, we transfer each of the 5 runs to the new task and once again report the mean and standard deviation.

Figure 2.11: The Rock-paper-scissors task requires jointly reasoning over the game observations and a document describing environment dynamics. The agent observes cell content in the form of text (shown in white).

Scenario	# graphs			# edges			# nodes		
	train	dev	unseen	train	dev	% new	train	dev	% new
permutation	30	30	y	20	20	n	60	60	n
new edge	20	20	y	48	36	y	17	13	n
new edge+nodes	60	60	y	20	20	y	5	5	y

Table 2.5: Statistics of the three variations of the Rock-paper-scissors task

2.10 Rock-paper-scissors

In addition to the main RTFM tasks, we also study a simpler formulation called Rock-paper-scissors that has a fixed goal. In Rock-paper-scissors, the agent must interpret a document that describes the environment dynamics in order to solve the task. Given an set of characters (e.g. a-z), we sample 3 characters and set up a rock-paper-scissors-like dependency graph between the characters (e.g. "a beats b, b beats c, c beats a"). We then spawn a monster in the world with a randomly assigned type (e.g. "b goblin"), as well as an item corresponding to each type (e.g. "a", "b", and "c"). The attributes of the agent, monster, and items are set up such that the player must obtain the correct item and then engage the monster in order to win. Any other sequence of actions (e.g. engaging the monster without the correct weapon) results in a loss. The winning policy should then be to first identify the type of monster present, then cross-reference the document to find which item defeats that type, then pick up the item, and finally engage the monster in combat. Figure 2.11

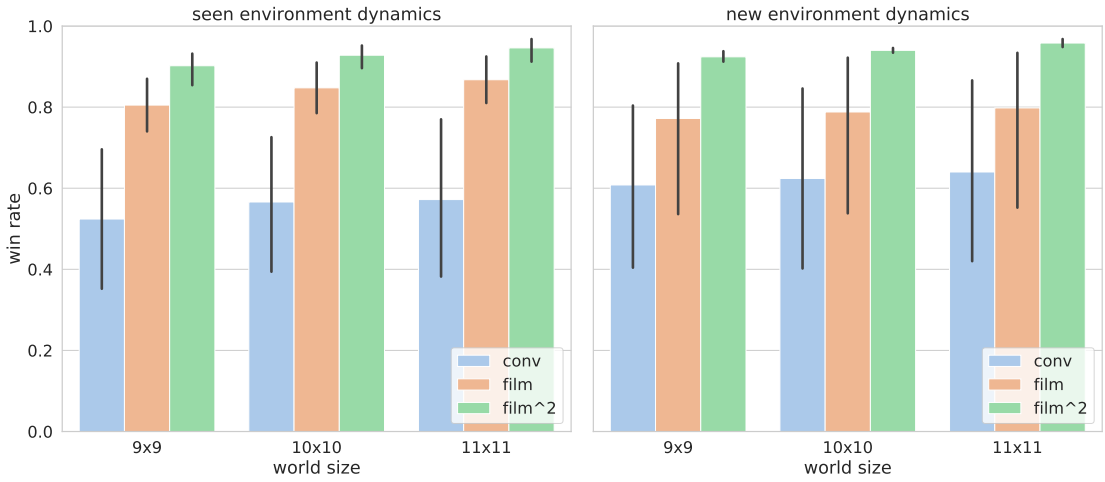


Figure 2.12: Performance on the Rock-paper-scissors task across models. Left shows final performance on environments whose goals and dynamics were seen during training. Right shows performance on the environments whose goals and dynamics were not seen during training.

shows an instance of Rock-paper-scissors.

Reading models generalise to new environments. We split environment dynamics by permuting 3-character dependency graphs from an alphabet, which we randomly split into training and held-out sets. This corresponds to the "permutations" setting in Table 2.5. We train models on the 10×10 worlds from the training set and evaluate them on both seen and not seen during training. The left of Figure 2.12 shows the performance of models on worlds of varying sizes with training environment dynamics. In this case, the dynamics (e.g. dependency graphs) were seen during training. For 9×9 and 11×11 worlds, the world configuration not seen during training. For 10×10 worlds, there is a 5% chance that the initial frame was seen during training.[2] Figure 2.12 shows the performance on held-out environments not seen during training. We see that all models generalise to environments not seen during training, both when the world configuration is not seen (left) and when the environment dynamics are not seen (right).

Reading models generalise to new concepts. In addition to splitting via permutations, we devise two additional ways of splitting environment dynamics by introducing new edges and nodes into the held-out

[2]There are 24360 unique grid configurations given a particular dependency graph, 4060 unique dependency graphs in the training set, and 50 million frames seen during training. After training, the model finishes an episode in approximately 10 frames. Hence the probability of seeing a redundant initial frame is $\frac{5e7/10}{24360*4060} = 5\%$.

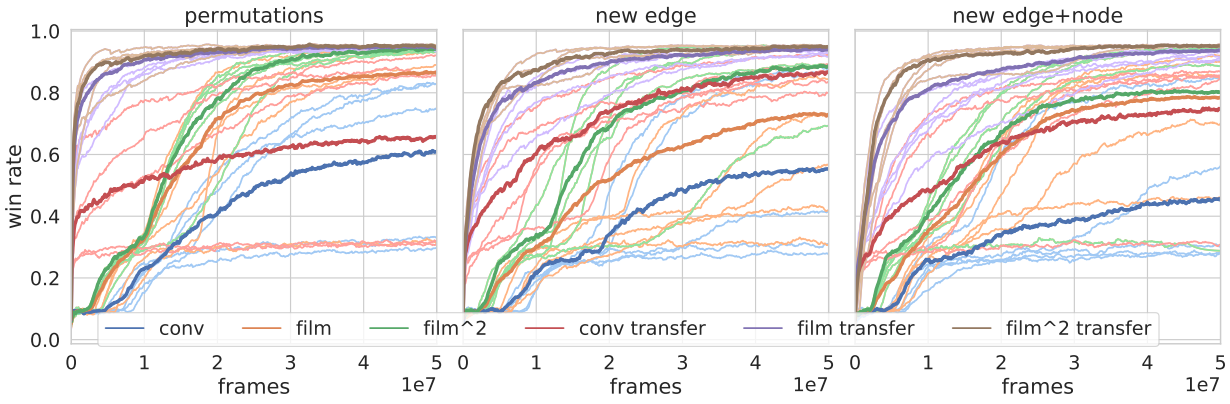


Figure 2.13: Learning curve while transferring to the development environments. Win rates of individual runs are shown in light colours. Average win rates are shown in bold, dark lines.

set. Table 2.5 shows the three different settings. For each, we study the transfer behaviour of models on new environments. Figure 2.13 shows the learning curve when training a model on the held-out environments directly and when transferring the model trained on train environments to held-out environments. We observe that all models are significantly more sample-efficient when transferring from training environments, despite the introduction of new edges and new nodes.

txt2π is more sample-efficient and learns better policies. In Figure 2.12, we see that the FiLM model outperforms the CNN model on both training environment dynamics and held-out environment dynamics. txt2π further outperforms FiLM, and does so more consistently in that the final performance has less variance. This behaviour is also observed in the in Figure 2.13. When training on the held-out set without transferring, txt2π is more sample efficient than FiLM and the CNN model, and achieves higher win-rate. When transferring to the held-out set, txt2π remains more sample efficient than the other models.

2.11 Curriculum learning training curves

2.12 Entities and modifiers

Below is a list of entities and modifiers contained in RTFM:

Monsters: wolf, jaguar, panther, goblin, bat, imp, shaman, ghost, zombie

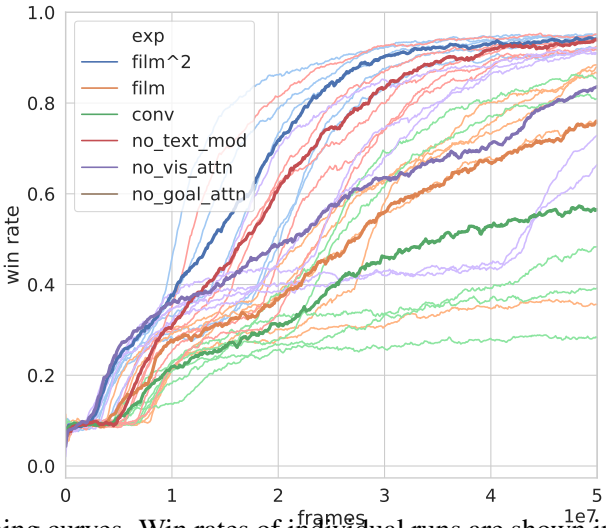


Figure 2.14: Ablation training curves. Win rates of individual runs are shown in light colours. Average win rates are shown in bold, dark lines.

Weapons: sword, axe, morningstar, polearm, knife, katana, cutlass, spear

Elements: cold, fire, lightning, poison

Modifiers: Grandmaster's, blessed, shimmering, gleaming, fanatical, mysterious, Soldier's, arcane

Teams: Star Alliance, Order of the Forest, Rebel Enclave

2.13 Language templates

We collect human-written natural language templates for the goal and the dynamics. The goal statements in RTFM describe which team the agent should defeat. We collect 12 language templates for goal statements. The document of environment dynamics consists of two types of statements. The first type describes which monsters are assigned to with team. The second type describes which modifiers, which describe items, are effective against which element types, which are associated with monsters. We collection 10 language templates for each type of statements. The entire document is composed from statements, which are randomly shuffled. We randomly sample a template for each statement, which we fill with the monsters and team for the first type and modifiers and element for the second type.

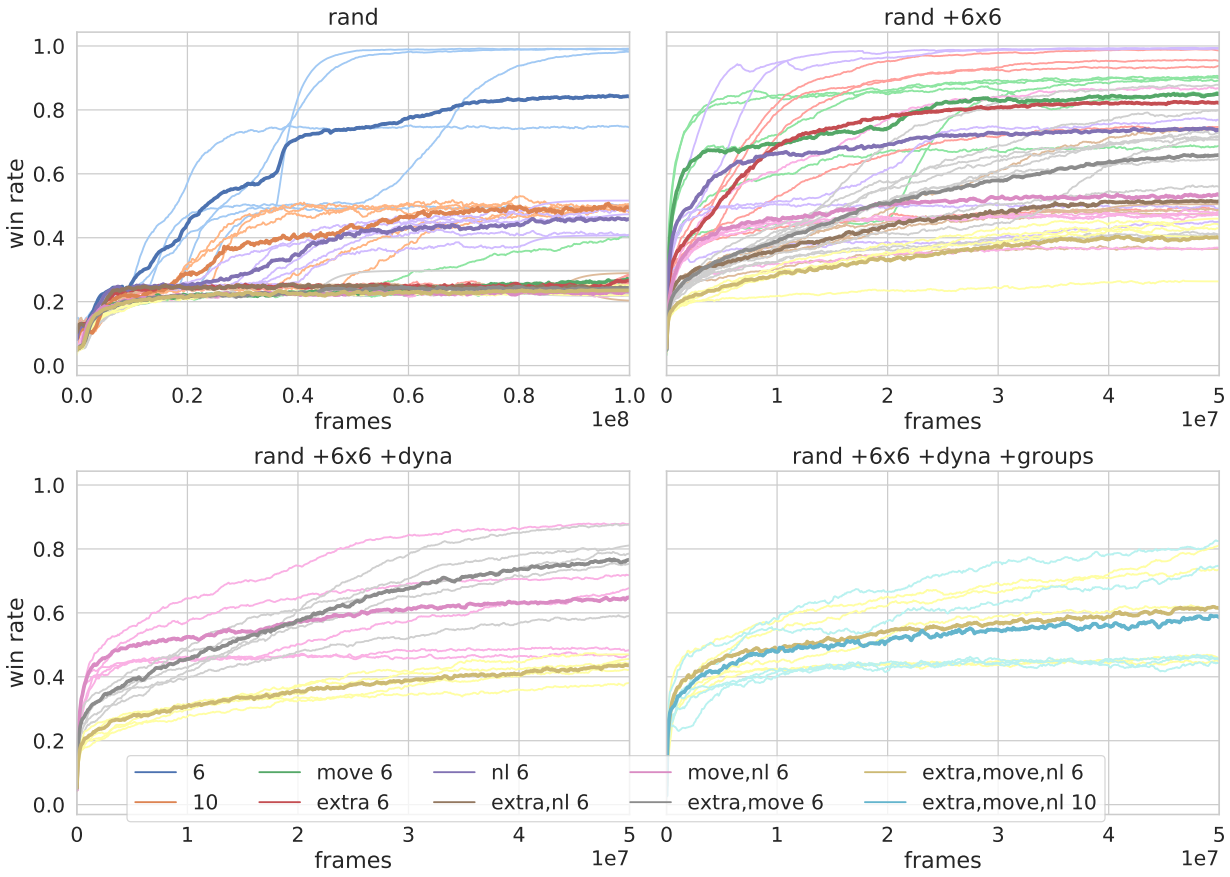


Figure 2.15: Curriculum learning results for txt2π on RTFM. Win rates of individual runs are shown in light colours. Average win rates are shown in bold, dark lines.

2.14 Conclusion

We proposed RTFM, a grounded policy learning problem in which the agent must jointly reason over a language goal, relevant dynamics specified in a document, and environment observations. In order to study RTFM, we procedurally generated a combinatorially large number of environment dynamics such that the model cannot memorise a set of environment dynamics and must instead generalise via reading. We proposed txt2π, a model that captures three-way interactions between the goal, document, and observations, and that generalises to new environments with dynamics not seen during training. txt2π outperforms baselines such as FiLM and language-conditioned CNNs. Through curriculum learning, txt2π performs well on complex RTFM tasks that require several reasoning and coreference steps with natural language templated goals and descriptions of the dynamics. Our work suggests that language understanding via reading

is a promising way to learn policies that generalise to new environments. Despite curriculum learning, our best models trail performance of human players, suggesting that there is ample room for improvement in grounded policy learning on complex RTFM problems. In addition to jointly learning policies based on external documentation and language goals, we are interested in exploring how to use supporting evidence in external documentation to reason about plans [Andreas et al., 2018] and induce hierarchical policies [Hu et al., 2019; Jiang et al., 2019].

Chapter 3

General-purpose Reading to Learn models

In the previous section, we looked at an example of Reading to Learn in generalizing to new RTFM games by reading manuals that describe game dynamics. In addition to playing new games by reading game manuals [Zhong et al., 2020b; Hanjie et al., 2021a], we can also consider interacting in new kitchens and navigating new scenes by reading scene descriptions [Shridhar et al., 2020; Chen et al., 2018]. Reading to Learn also applies to very different problems such as conversing about new topics by reading domain FAQs [Zhong & Zettlemoyer, 2019] and translating language to code for new databases by reading database descriptions [Zhong et al., 2020a]. In prior work, practitioners build environment-specific models that addressed the unique challenges in each environment. How can we build general Reading to Learn models that work across different language grounding challenges, as Figure 3.1 illustrates?

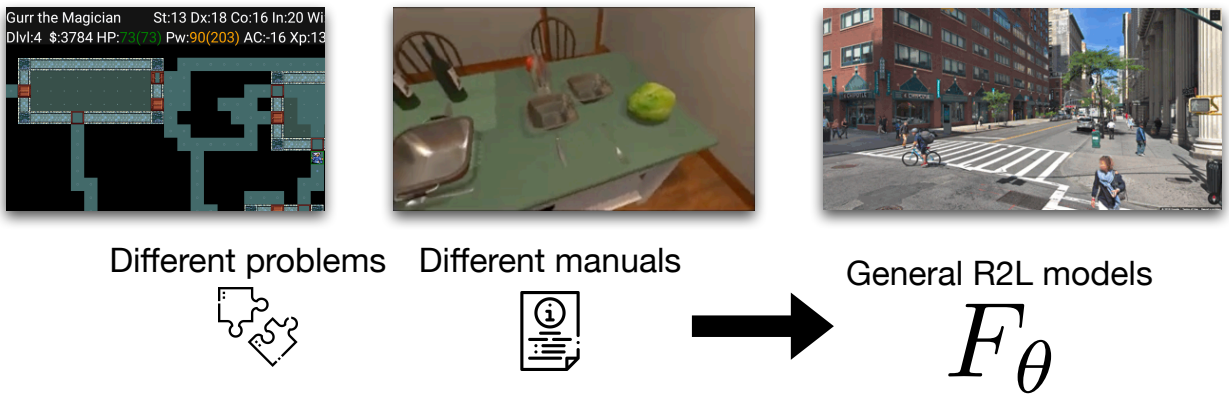
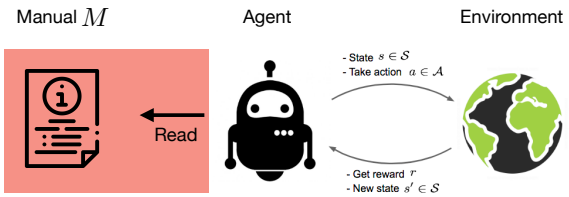
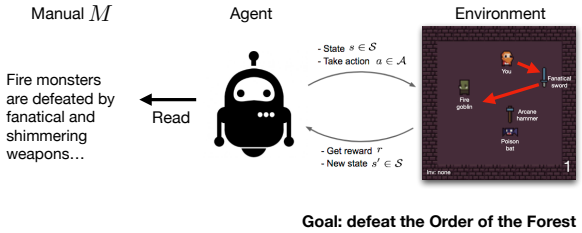


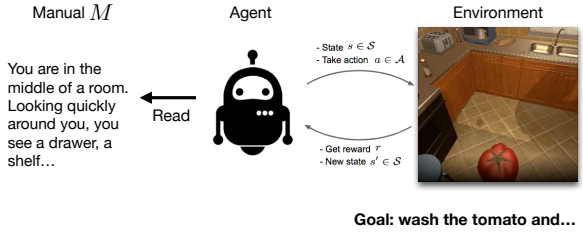
Figure 3.1: Building general Reading to Learn models



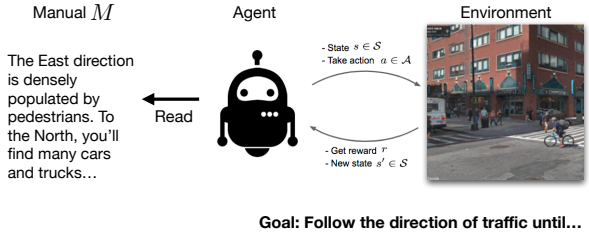
(a) Generalizing to new environments by reading manuals.



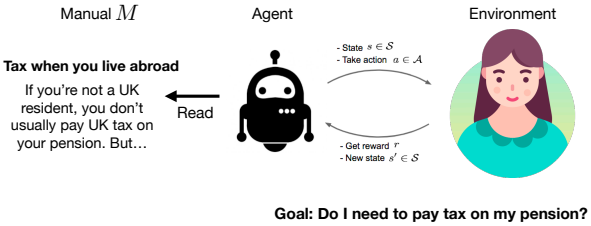
(b) Playing new games by reading game manuals.



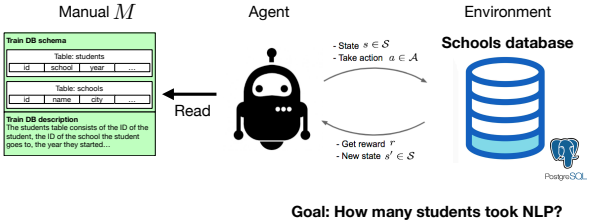
(c) Interacting with new kitchens by reading room descriptions.



(d) Navigating new neighbourhoods by reading neighbourhood descriptions.



(e) Conversing about new topics by reading domain FAQs.



(f) Translating language questions to code for new DBs by reading DB schema and descriptions.

Figure 3.2: Applications of Reading to Learn.

In this section, we will broaden the scope of Reading to Learn by building a benchmark for Reading to Learning that tests for different language-grounding challenges. We will then look at extending txt2π to obtain a general-purpose Reading to Learn model for learning policies in grounded environments. Figure 3.2 shows several examples of Reading to Learn applications.

3.1 Introduction

An ideal language-conditioned agent should interpret language in diverse environments with varying observation space, action space, language, and plan complexity. However, existing language-grounding literature typically focuses on single environments, and proposes methodological contributions specific to those environments [Luketina et al., 2019; Tellex et al., 2020]. In order to determine which contributions are environment-specific and which apply across multiple environments, it is critical to develop universal models that can be easily evaluated in many different settings.

To facilitate this research, we present the multi-environment Symbolic Interactive Language Grounding Benchmark (SILG). We focus on symbolic environments with semantic symbols instead of raw visual observations for efficiency, interpretability, and emphasis on abstractions over perception. SILG consists of diverse environments including grid-worlds RTFM [Zhong et al., 2020b], Messenger [Hanjie et al., 2021b], and NetHack [Küttler et al., 2020], which require generalization to new dynamics (i.e. how entities behave), entity references, and partially observed worlds. SILG also contains symbolic counterparts of visual grounding environments ALFRED [Shridhar et al., 2020] and Touchdown [Chen et al., 2018] , which require interpreting rich natural language in complex scenes. For the former, we use its textual variant ALFWorld [Shridhar et al., 2021]. For the latter, we create SymTD by applying object segmentation to Touchdown panoramas. Despite significant implementation differences, we unify these environments under a common interface in SILG, so that one can easily develop and evaluate language grounded RL methods across all of these challenges.

SILG environments present a variety of unique grounding challenges in the richness of the observation space, action space, language specification, and plan complexity. We quantify these challenges and additionally analyze the success rate and lengths of expert playthroughs. For visual grounding environments, we show symbolic variants (ALFWorld and SymTD) facilitate faster learning and result in policies that transfer

to their visual counterparts. While a unified model may not outperform specialized models engineered for specific environments, it can be helpful to understand whether particular modelling innovations are environment specific or more general techniques. Furthermore, while the challenges in each environment are very different, we want to encourage the development of unified architectures and approaches that can scale across many language grounding tasks.

In addition to SILG, we propose the Symbolic Interactive Reader (SIR), the first shared model architecture for these environments. We combine SIR with several recent advances in language-conditioned RL, including FiLM^2 [Zhong et al., 2020b], egocentric local convolution [Hill et al., 2020a], recurrent state-tracking [Küttler et al., 2020], entity-centric attention [Hanjie et al., 2021b], and large pretrained LMs [Hill et al., 2020b]. On most environments, SIR achieves comparable performance to methods designed specifically for single environments. In addition, we find that many recent advances do not result in significant gains on environments other than the one they were designed for. This highlights the need for a multi-environment benchmark. Finally, the best models significantly underperform humans on SILG (10-85% depending on environment), which suggests ample room for modelling improvements that generalize across environments.

In summary, we (1) combine five language-grounding environments under the same interface to evaluate language grounded RL methods across diverse grounding challenges, (2) present the first shared model architecture for these environments, and (3) analyze recent modelling contributions across these environments. We hope SILG enables the community to quickly identify new models and learning algorithms that generalize to a diverse set of environments and their associated challenges. The code for SILG is available at `https://github.com/vzhong/silg`.

3.2 SILG Environments

SILG contains five language-grounding environments including both grid-worlds (RTFM, Messenger, SILGNetHack) and symbolic counterparts of 3D-visual worlds (ALFWorld, SymTD). While all involve agents situated in interactive worlds, each presents unique challenges in richness of observation space, action space, language specification, and plan complexity. Table 3.1 quantifies their theoretical complexity along these

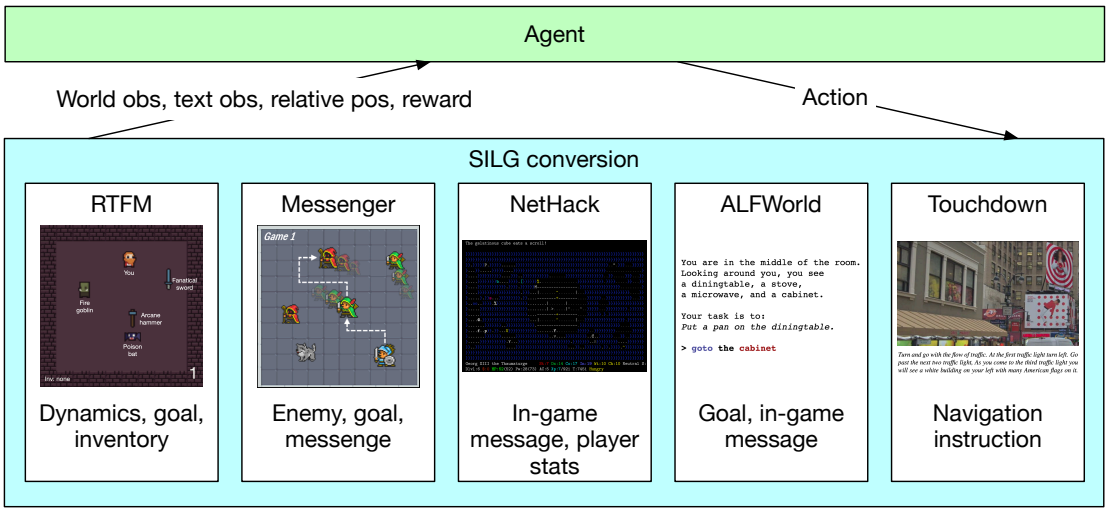


Figure 3.3: Environments included in SILG. The world observations and text fields are shown for each environment. Detailed examples are in Section 3.10.

dimensions as well as empirical complexity using expert playthroughs.[1]

The goal of SILG is to provide a simple-to-use benchmark that allows researchers to quickly evaluate methods across all of these environments as well as their respective challenges. We thus combine these environments under a unified interface built on top of OpenAI Gym [Brockman et al., 2016]. In each environment instance, the agent observes text inputs as well as world observations. For grid worlds such as RTFM, Messenger, and SILGNetHack, the agent receives a 2-D bird's-eye-view symbolic grid as observations. For visually inspired environments such as ALFWorld and SymTD, the agent receives a symbolic egocentric view of the present scene. Figure 3.4 shows how SILG environments are rendered to players via the `play` utility. In the rest of this section, we describe each SILG environment in detail. Section 3.6 shows how to use SILG in Python. Section 3.11 shows licensing for SILG environments.

Selection criteria We select interactive environments that span the challenges presented in Table 3.1, are easily converted to symbolic representations, and avoid the use of additional simulators (e.g. Matterport3D [Anderson et al., 2018a]). While visual perception is clearly important for language grounding [Ferraro et al., 2015], we focus on the unique challenges of symbolic environments such as multi-hop reasoning and generalization to rich sets of procedurally generated dynamics. We leave the challenge of developing a visually rich multi-environment grounding benchmark to future work.

Due to the lack of gold trajectories in many of the selected environments, we do not support imitation

[1]For each environment, an expert plays as many episodes as necessary to learn about the game. We then record the playthroughs to compute the empirical win rate and trajectory length. More details in Section 3.10.

Table 3.1: SILG statistics. "dynamics" are high level rules dictating behaviour of entities. "Ref hops" are number of intra-text references the agent must resolve to determine correct course of action. Messenger and SymTD text are human-written instead of procedurally generated. Distinctive properties are **bold**.

	RTFM	Messenger	SILGNetHack	ALFWorld	SymTD
Action space	5 fixed	5 fixed	23 fixed	**50+ choices**	1-5 choices
State space	6×6 grid 5 entities	10×10 grid 14 entities	21×79 **partial obs**	102 nodes 191 entities	**29.6k complex panoramas**
Mean text len	31 words	30 words	9 words	**100 words**	**90 words**
Vocab size	262 words	595 words	$\sim$100 words	1237 words	**4999 words**
Generalization	**new dynamics**	**new dynamics**	new layouts	new instr new layouts	new instr
Ref hops	**6 hops**	3 hops	1 hop	$\sim$4 hops	**$\sim$7 hops**
Human win %	100%	100%	78.1%	100% new instr 100% +layouts	61.5%
Human # steps	6.0 steps	2.2 steps	34.4 steps	7.8 steps new instr 9.6 steps +layouts	33.6 steps
Env FPS	240	1627	439	7	779
Key challenge	multi-step reasoning	adversarial generalization	partial obs	large action space	complex language

learning (IL) in this version of SILG.

RTFM RTFM [Zhong et al., 2020b] is a grid-world environment where an agent interprets text to acquire the correct items to fight the correct monsters. A key challenge in RTFM is multi-modal multi-step reasoning (at least 6 steps) combining world observations with texts associated with multiple entities. Given a team to beat, the agent must identify which monster is on the team, then identify the item descriptor that would beat the monster descriptor. Finally, the agent must acquire the item with the correct descriptor and engage the correct monster to win. RTFM evaluation is on games with unseen rules, forcing agents to make novel reasoning steps to generalize successfully. At each step, the agent receives a symbolic grid containing names of entities present, as well as texts indicating the high level rules, the agent inventory, and the goal of the particular game instance. We include all 4 RTFM curriculum stages, but only show results for the first stage in this preliminary study.

Messenger Messenger [Hanjie et al., 2021b], is a grid environment where the agent must acquire a message and deliver it to the goal while avoiding an enemy after extracting entity-role assignments from a text manual. A key challenge in Messenger is the adversarial train-evaluation split without prior entity-text

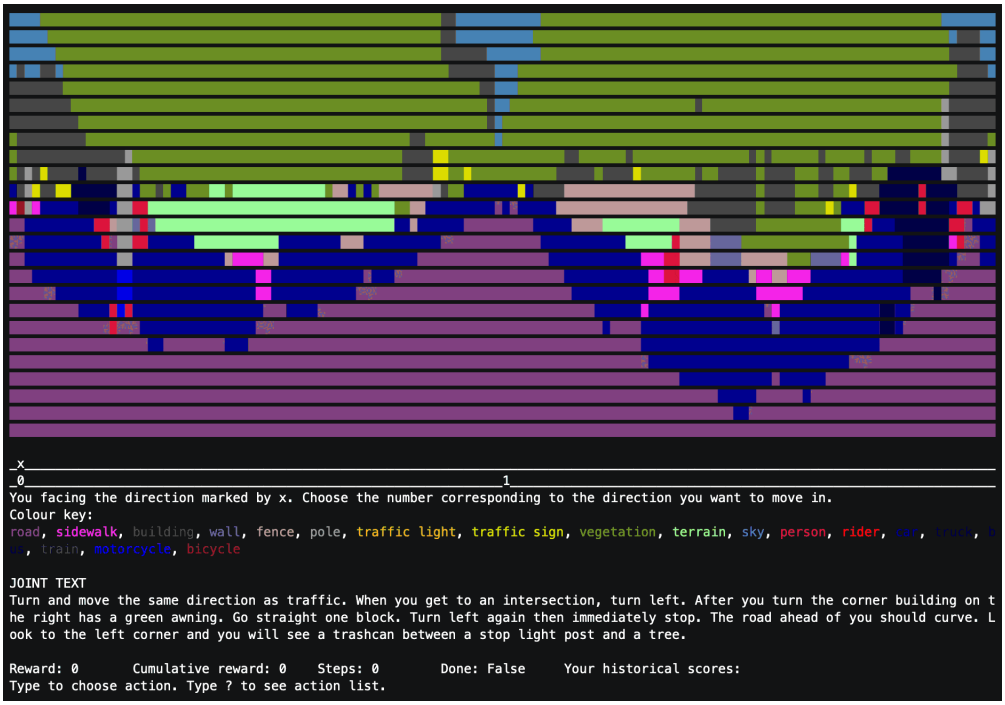


Figure 3.4: The SILG `play` utility (shown here for SymTD) enables human playthrough as well as visualizing what input the model observes. Because all environments are symbolic, the `play`utility works in console (e.g. via `ssh`, `tmux`) without need for X-forwarding.

grounding. There is no overlap in entity-role assignments between training and evaluation, forcing agents to make compositional entity-role generalizations. At each step, the agent receives a symbolic grid containing symbol IDs of entities present, as well as texts indicating roles of each entity. The entities are referred to in text by many names, which have no lexical overlap with their symbol ID. That is, the text "dog" in the text for example is the non-textual symbol 2 in the observation and the association between entities and references must be learned via interaction. We include all 3 Messenger curriculum stages, but only show results for the first stage in this preliminary study.

SILGNetHack NetHack is a a complex rogue-like game from the NetHack learning environment [Küttler et al., 2020]. In SILGNethack, we combine 3 tasks (`Score`, `Gold`, and `Scout`) and specify the task to complete for each episode via a text prompt. SILGNetHack is challenging due to its large state space and partial observability. The agent may descend multiple floors and sections of each floor may be obscured until exploration by the agent. Because of the different score distributions of each task, we mark a trajectory as successful if it exceeds a task-specific score threshold determined from human playthroughs. We evaluate agents on previously unseen map layouts that are procedurally generated with new seeds disjoint from the

ones used during training. More information about the SILG multi-task SILGNetHack is in Section 3.8. At each step, the agent receives a symbolic grid containing symbol IDs of entities present, as well texts denoting the goal, agent stats, and feedback from the environment after the agent's last action. SILGNetHack vocabulary is technically infinite because players can arbitrarily name things, however in our expert playthroughs of SILG SILGNetHack, we observe just over 100 unique words. Human experts win just under 80% of games with an average of 34 steps, which demonstrates the challenge of SILGNetHack. All failures can be attributed to hitting the step limit before acquiring the necessary win conditions.

ALFWORLD (text ALFRED) In ALFWorld, an agent navigates and manipulates objects inside a 3D kitchen [Shridhar et al., 2021]. Its large text action space, with more than 50 valid actions (given by the game engine) for most scenes is a key challenge. Unlike its visual counterpart ALFRED [Shridhar et al., 2020] where the agent observes 3-D images of the kitchen, in ALFWorld the agent must rely on language descriptions of the kitchen. Goals are provided in human written language (e.g. put a clean sponge on the metal rack). The language in ALFWORLD is not complex, but are 100 words on average due to a large number of items in a single scene. Following recent work [Shridhar et al., 2021], we evaluate on both unseen instructions (new instr) and unseen room layouts (new layouts). At each step, the agent receives the goal text and a list of items present in the room (e.g. "cup 1", "bottle 2"). We concatenate the names of these items into a symbolic world observation grid, each entry containing the name of one item. The agent then selects from plausible commands given what is present in the scene.

SILGTouchdown (SymTD, VisTD) In Touchdown, the agent navigates through Google Street View panoramas according to long compositional instructions that tests spatial reasoning [Chen et al., 2018; Mirowski et al., 2018; Mehta et al., 2020]. A key challenge is the rich human-written navigation instructions that describe photorealistic images. Touchdown's long human-written instructions contain many intra-text reference hops, which we approximate as the number of sentences plus the number of sequential connectors such as "then". We convert Touchdown to a symbolic environment by segmentating its panoramas into semantic grids. In each step, the agent observes the instruction text and a grid of discretized segmentation class IDs corresponding to the current panorama. It then chooses among a list of radial directions to proceed to the next panorama. The agent wins if it passes the goal location. We use the same train-test split as the original Touchdown environment, which features unseen navigation texts.

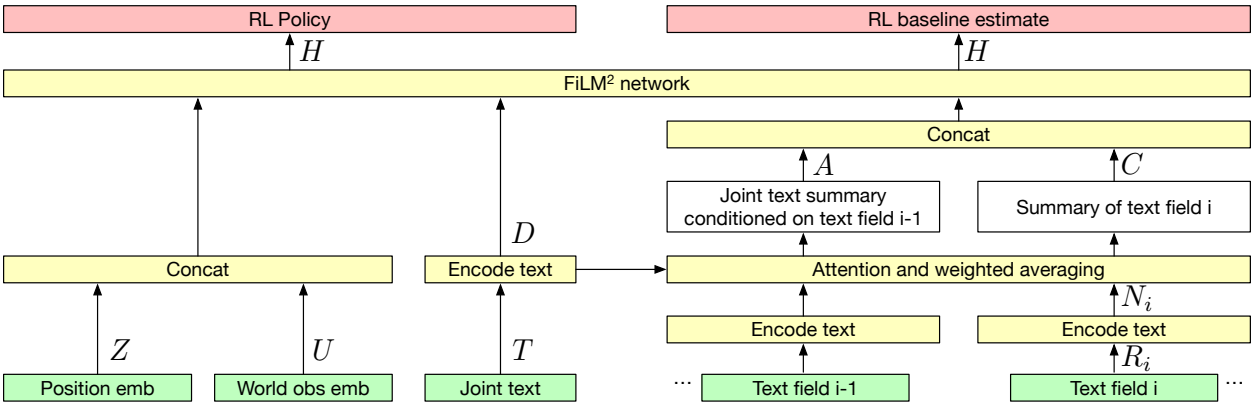


Figure 3.5: The Symbolic Interactive Reader (SIR) baseline. Inputs are green, intermediate results white, outputs red, and model components yellow. Details about the FiLM^2 layer is in Section 3.7.

We show that our symbolic Touchdown (SymTD) facilitates faster learning compared to learning in its visual equivalent (VisTD). Human performance demonstrates some limitations of SymTD, with an expert win rate just over 60%. This may be due to the symbolic representations removing information referenced by the instructions such as color, or because the segmented features are visually disparate from real-world views [Dubey et al., 2018]. We also include manual stop variants of SymTD and VisTD, which are functionally equivalent to the original Touchdown. Section 3.9 details these variants, SymTD/VisTD creation as well as discussions on human performance. Compared to prior work on Touchdown and ALFWorld, we train using RL without supervised trajectories as opposed to imitation learning.

3.3 The Symbolic Interactive Reader Baseline Model

Figure 3.5 shows the SIR baseline for the SILG benchmark. To the best of our knowledge, this is the first shared model architecture for RTFM, Messenger, NetHack, ALFWorld, and Touchdown. Consider an agent situated in an arbitrary SILG environment. At each time step t, the model receives from the environment the following inputs (precise inputs for each environments are shown in Section 3.10).

- **World observations** $X \in \mathbb{R}^{h \times w \times k}$ where h and w are the height and width of the observation and each element corresponds to the k-word symbol ID(s) of its content.
- **Joint text** $T \in \mathbb{R}^{l}$ of l tokens of the text to attend over.
- **Text fields** $R \in \mathbb{R}^{n \times m}$ where the ith row contains the ith of n environment text field such as agent

inventory or environment feedback. m is the max token count of these texts.

- **Relative position** $Z \in \mathbb{R}^{h \times w \times 2}$ cell-wise feature that denotes the position of each cell relative to the player agent in the x and y directions.

As a policy learner, the model must output a distribution Y over the action space. We additionally output a baseline estimate of the value function to stabilize policy learning [Espeholt et al., 2018a]. Let d and r denote embedding and bidirectional LSTM sizes. We first sum embeddings for each cell in the world observation to obtain world representation $U = \mathrm{sum}\left(\mathrm{emb}\left(X\right)\right) \in \mathbb{R}^{h \times w \times d}$. Next, we encode the ith text field R_i and the joint text T using an bidirectional LSTMs [Hochreiter & Schmidhuber, 1997].

$$N_i = \mathrm{BiLSTM}N\mathrm{emb}\left(R_i\right) \in \mathbb{R}^{m \times r} \tag{3.1}$$

$$D = \mathrm{BiLSTM}D\mathrm{emb}\left(T\right) \in \mathbb{R}^{l \times r} \tag{3.2}$$

We then compute weighted average over text fields $\tilde{C}_i$ and attention $\tilde{A}_i$ over the joint text.

$$\tilde{C}_i = \mathrm{selfattn}iN_i = \sum_j \mathrm{Softmaxlinear}_i\left(N_i\right)_j N_{ij} \in \mathbb{R}^r \tag{3.3}$$

$$\tilde{A}_i = \mathrm{attend}\left(D, \tilde{C}_i\right) = \sum_j \mathrm{Softmax}D\tilde{C}_{ij} D_j \in \mathbb{R}^r \tag{3.4}$$

We compress $\tilde{C} \in \mathbb{R}^{n \times r}$ and $\tilde{A} \in \mathbb{R}^{n \times r}$ again to support any number of text fields.

$$C = \mathrm{selfattn}C\tilde{C} \in \mathbb{R}^r \tag{3.5}$$

$$A = \mathrm{selfattn}A\tilde{A} \in \mathbb{R}^r \tag{3.6}$$

We now have representations for world observations U, text fields C, and joint text conditioned on text fields A. We apply successive FiLM^2 layers to build multiple levels of codependent representations between texts and world observations to model multiple cross-modal reasoning steps [Zhong et al., 2020b]. To support arbitrary number of text fields, we modify the text input of the ith FiLM^2 layer to be the concatenation of the text fields C, attention over joint text conditioned on text fields A, and attention over joint text conditioned on the visual summary of the last FiLM^2 layer $s^{(i-1)}$.

$$V^{(i)}, s^{(i)} = \mathrm{FiLM}^2\left([V^{(i-1)}; Z], [C, A, \mathrm{attend}\left(D, s^{(i-1)}\right)]\right) \tag{3.7}$$

We use the definition of FiLM$^2(\text{visuals}, \text{texts})$ from Zhong et al. [2020b]. We define $V^{(1)}$ and $s^{(1)}$ to be the initial world observation U and its spacial max-pooling. Finally, we use a multi-layer perceptron to build a fixed-size codependent representation of the inputs based on the last FiLM^2 layer's output $H = \tanh\left(\text{linear}_4\left(\text{flatten}(V^{(\text{last})})\right)\right)$, which is used to compute the baseline estimate of the value function $y_{\text{baseline}} = \text{MLP}y_{\text{baseline}}H$ and the policy $Y(H)$ expressed as a distribution over actions. While the core architecture of SIR is identical for all environments, a different policy module Y is necessary for different types of action spaces.

Fixed sized action space (RTFM, Messenger, SILGNetHack) We simply apply a multilayer perceptron to the final representation $Y = \text{MLP}YH$.

Multiple-choice text action space (ALFWorld) Let Q_j denote tokens for the jth choice (e.g. pick up the mug), which we encode a bidrectional LSTM $G_j = \text{BiLSTM}G\text{emb}\left(Q_j\right)$. We then attend over this text using the final representation H to score for jth choice $Y_j = \text{linear}_4\left(\text{attend}\left(G_j, H\right)\right)$.

Multiple-choice navigation action space (SILGTouchdown) Let j denote the index of the world representation corresponding to a movement direction. For example, for a world observation width of 100, the index corresponding to advancing in the 30 degrees direction is $\frac{30*100}{360} \approx 8$. We encode the navigation choice by selecting its corresponding world observation representation, then scoring it via dot product with the final output representation $Y_j = \text{linear}_5\left(U_j\right)^\intercal H$.

3.4 Experiments

Setup How well does a shared architecture do across all five SILG environments? To answer this, we train and evaluate SIR using Torchbeast [Küttler et al., 2019b], a distributed RL framework with importance weighted actor-learners based on IMPALA [Espeholt et al., 2018a]. For each environment (separately), we train on training, do early stop on validation, and evaluate on test. NetHack does not distinguish between train and evaluation, hence we create our own splits by dividing the seed range (first 1 million seeds for training, second for validation, and third for test). We run 5 random seeds for each environment. The hyperparameter and compute resources are respectively shown in Section 3.12 and 3.13. SILG.

Table 3.2: Success rate on test environments for SIR and its best variant. Standard deviation are in brackets. We early stop on validation and evaluate best checkpoint on test. For RTFM, Messenger, and SILGNetHack, we evaluate 100 episodes. For ALFWorld and Touchdown, we evaluate on initial states from each test episode. The variant with best performance across envs is `+state`. The SOTA for RTFM, Messenger, and ALFWorld are respectively from Zhong et al. [2020b], Hanjie et al. [2021b], and Shridhar et al. [2021] (std was not reported in ALFWorld). $^{\triangle}$SOTA for ALFWorld relies on supervised trajectories and beam search, which SIR does not use. There are no previous results for multitask SILGNetHack and SymTD as they are introduced here. Though not comparable, the manual stop VisTD SOTA trained using imitation learning on supervised trajectories is 16.7% [Xiang et al., 2020].

Model	RTFM	Messenger	SILGNetHack	ALFWorld		SymTD
				new inst	new inst+layouts	
Base	88.8 (22.4)	0 (0)	23.8 (0.8)	21.0 (1.5)	16.0 (2.1)	9.7 (1.3)
Best	`+state`	`+all`	`+local conv`	`+state`	`+state`	`+state`
	99.2 (0.7)	31 (2.6)	25.4 (3.3)	23.6 (2.8)	16.6 (2.9)	14.9 (1.8)
SOTA	83 (21)	85 (1.4)	N/A	40$^{\triangle}$	37$^{\triangle}$	N/A
Human	100	100	78.1	100	100	61.5

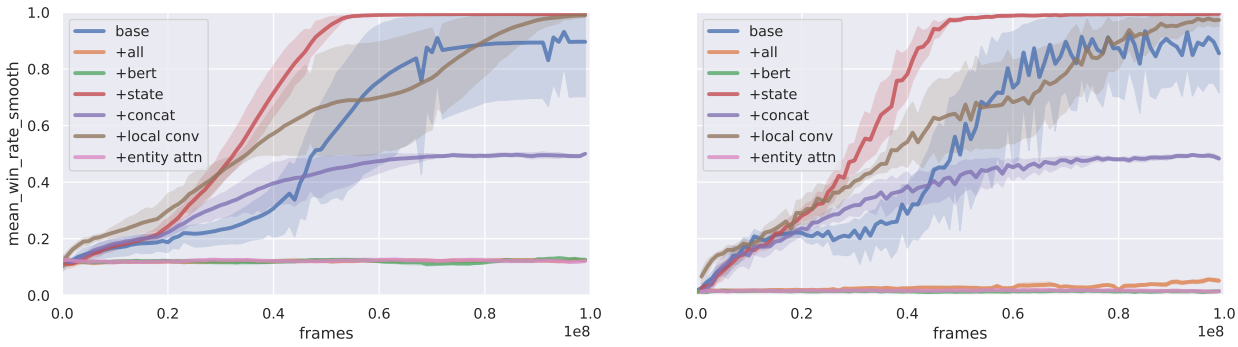


Figure 3.6: RTFM performance. Left: train envs, right: validation envs.

Results Figures 3.6 through 3.10 show learning curves for each environment. Table 3.2 shows the test performance for the baseline model and the best model variant. Despite sharing the same core model architecture, SIR achieves reasonable performance across all environments except Messenger, where it overfits due to lack of pretrained LM and entity-centric attention. Nevertheless, the best performing model significantly trails human performance, indicating room for further improvement.

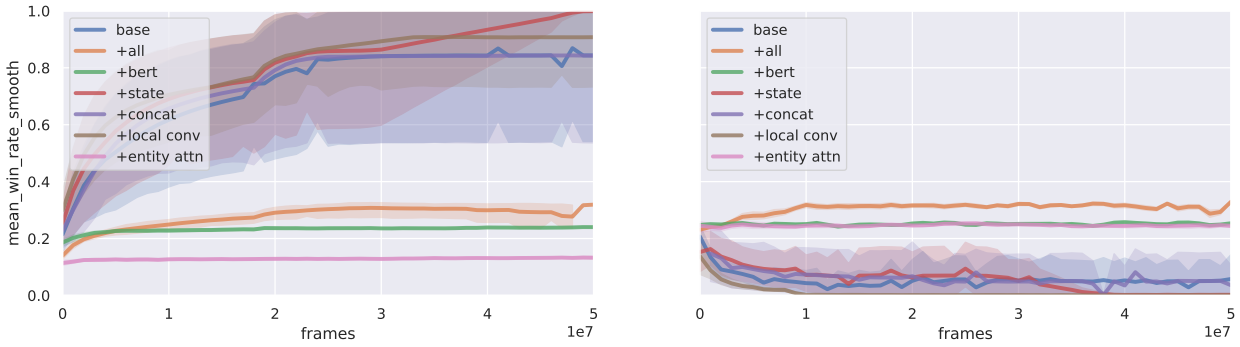


Figure 3.7: Messenger performance. Left: train envs, right: validation envs.

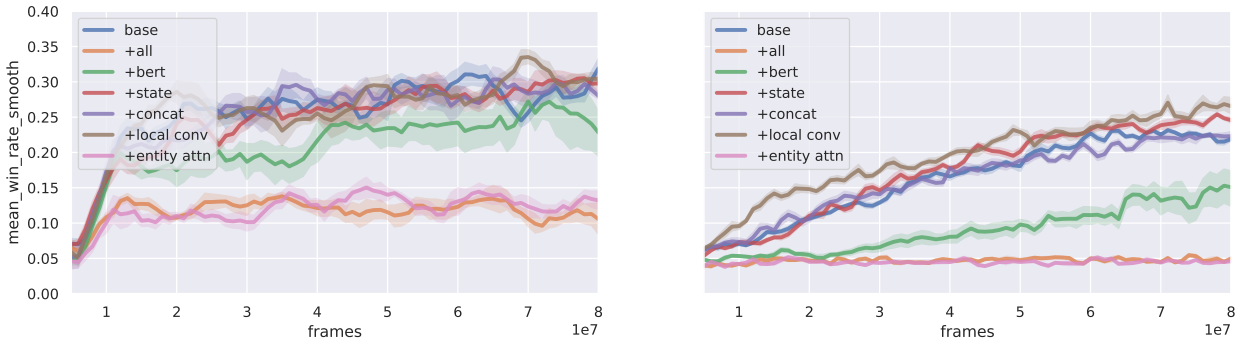


Figure 3.8: SILGNetHack performance. Left: train envs, right: validation envs.

3.4.1 Analyses of recent grounded language RL modelling contributions

Next, we use SILG to evaluate recent modelling advances for language grounding across environments by adding them to the SIR baseline. These modelling enhancements were proposed for (and resulted in key gains on) the environments included in SILG. Namely, we analyze the effectiveness of recurrent state-tracking, entity-centric local convolution, entity-centric attention, and pretrained LMs.

Recurrent state tracking (`state`) As in Küttler et al. [2020], we augment the SIR baseline with a state-tracking LSTM by replacing the final H with $H' = H + \text{LSTM}(H, S_{t-1})$, where S_{t-1} is the previous LSTM state (summing LSTM output and H outperforms replacing H with LSTM output). State-tracking consistently improves convergence and generalization, even when the correct next step is fully determined by current world observations (e.g. RTFM). This may be because it helps prevent local minima that cause repetitive actions. The exception to this is Messenger, where state-tracking does not help generalize to the evaluation distribution.

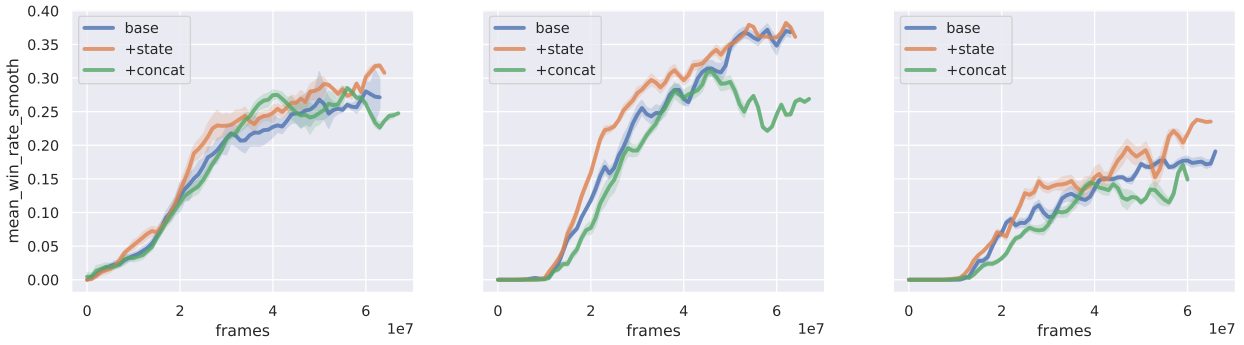


Figure 3.9: ALFWorld perfomance. Left: train envs, middle: new instruction validation envs, right: new instruction+new layouts validation envs. For efficiency we only evaluate on a subset (50 out of 140) of the validation environments for early stopping. We do train BERT variants here due to computational constraints. ALFWorld does not have entity IDs and no agent location, hence we do not show local convolution nor entity attention experiments.

Entity-centric local convolution (`local conv`) Hill et al. [2020a] proposed local convolution around the agent to obtain an egocentric view of world observations. While this helps generalize in SILGNetHack, it does not help significantly in other environments. One reason is that this provides redundant information as positional embeddings, which is already included in the base model and is a cheaper alternative to adding an additional egocentric convnet.

Entity-centric attention (`entity attn`) Hanjie et al. [2021b] propose replacing entity representations with attention over text specification, such that the world observations are forcibly composed using text representations. We add this by replacing world representation U with entity attention over text fields R as described in Hanjie et al. [2021b]. This constraint causes underfitting of SIR on most environments. Since the entity representation is built entirely using the text, when there is incomplete entity information or it is difficult to extract the relevant information from the manual text this can be a handicap. However, for Messenger, entity-centric attention prevents overfitting.

Pretrained language model (`bert`) A natural question in language-grounding is how to leverage large, pretrained LMs [Hill et al., 2020b]. We use a simple method to incorporate BERT [Devlin et al., 2019] by replacing all text encoding with the summation of the original bidirectional LSTM encoding and BERT encoding. Due to the memory requirement of large pretrained LMs, we cannot fine-tune the LM during training, and thus keep the LM parameters fixed. Pretrained LMs (`bert` and `all`) help generalization in

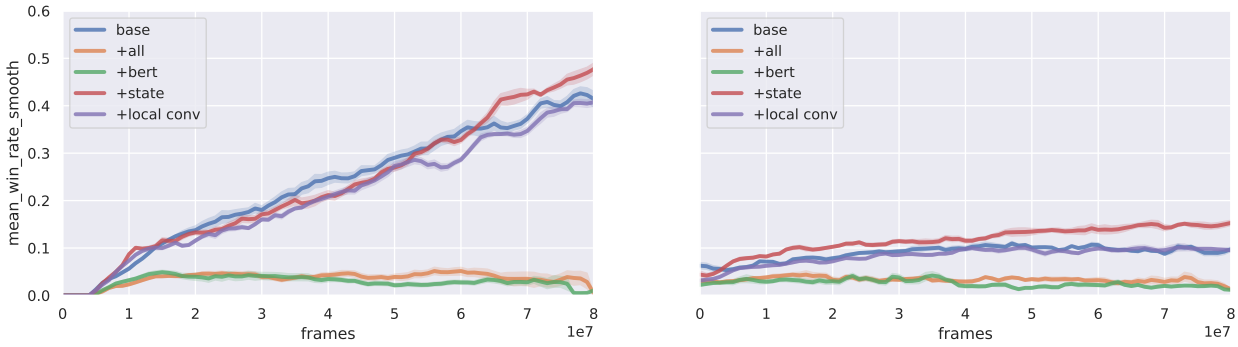


Figure 3.10: SymTD performance. Left: train envs, right: validation envs. Touchdown does not have entities, hence we do not show experiments for entity attention.

Messenger but does not improve performance on other environment in our experiments. For tasks such as RTFM and SILGNetHack, our use of a general-purpose LM may not be beneficial for the highly specific language used in those tasks (i.e. fantasy world with word like shaman, goblin, mage etc). We stress that this is a preliminary investigation into the use of LMs on these environments, and we encourage future research on how to effectively use pretrained LMs across environments using SILG.

3.4.2 Analyses of SILG environments

Finally, we examine performance of SIR and variants to analyze challenges presented by SILG.

Generalization requirement of environments SILG's evaluation environments require different types of generalization. RTFM requires generalizing to new environment dynamics by referring between world observations and multiple texts; because SIR adopts FiLM^2 from Zhong et al. [2020b], it is able to achieve such generalization. Messenger requires compositional entity-role generalizations. That is, if an entity (e.g. dog) has a certain role (e.g. message holder) in training, such an entity-role assignment never appears in validation or test. SIR quickly overfits to to entity-role assumptions (e.g. dog as the message) in training suggesting the need for additional work on achieving this type of generalization using a joint model architecture. Combining pretrained LM with other enhancements (`+all`) results in generalization improvement, however the convergence remains very slow. This suggests that generalizing to new dynamics across environments without obvious lexical cues from the text remains a difficult challenge. SILGNetHack and ALFWorld require generalizing to new procedurally generated scenes, which SIR achieves. In the additional out-of-domain ALFWorld evaluation where the model must generalize to new layouts, state-tracking allows the model to

Table 3.3: Transfer task success rate from symbolic to visual envs for baseline and its best variant. Standard deviation shown in brackets. For ALFRED, we give ALFWorld-trained models language templates filled with detected objects from vision using oracle and Masked-RCNN object detectors.

Model	ALFWorld/ALFRED						SymTD	VisTD
	new inst			new inst + layouts			to	
	text	oracle	m-rcnn	text	oracle	m-rcnn	VisTD	
Base	21.0(1.5)	11.2(3.3)	3.0(1.7)	16.0(2.1)	0.7(0.4)	0.3(0.2)	9.7(1.3)	4.3(4.0)
Best	`+state`			`+state`			`+state`	`base`
	23.6(2.8)	11.3(1.9)	7.1(1.1)	16.6(2.9)	1.3(1.1)	0.7(0.6)	14.9(1.8)	4.3(4.0)

generalize faster. Touchdown requires generalizing to new natural language instructions. Here, the baseline suffers from a large generalization gap. We hypothesize that more effective means of incorporating pretrained LMs is necessary to achieve this type of generalization.

Necessity of separate text fields In `concat`, we concatenate text fields into a single string, which we encode using a bidirectional LSTM. In this case, both joint text D and text field representations N are set to this encoding. This degrades performance especially in RTFM, which shows that multi-hop references is more easily learned when the text fields are separated and modeled via structured attention. Note that this model variant is not shown for Touchdown because it only has one text field.

Learning from symbolic vs visual world observations Table 3.3 shows that policies learned in the symbolic environment transfer to the 3-D environment. Using oracle and Masked-RCNN [He et al., 2017], the ALFWorld policy can be transferred by filling observation text templates using detected objects. Our result with oracle detector is in line with Shridhar et al. [2021], though our performance is weaker because we do not use annotated data nor DAgger [Ross et al., 2011]. As with prior results, transfer to visual worlds with new layouts remains very challenging [Shridhar et al., 2021]. Transfer using Masked-RCNN results in large drop in performance, nevertheless SILG allows perception, albeit an important challenge, to be factored out so that one can focus and quickly iterate on abstraction challenges. Table 3.3 also shows that models trained on SymTD outperform those trained on VisTD (where U is 10-dim PCA features from a ResNet [He et al., 2016] panorama encoding) despite being faster (383 for SymTD vs. 344 frames per second for VisTD). That is, by applying segmentation to obtain SymTD, we are able to obtain a better policy than training directly with visual features using VisTD. The results from both ALFWorld and SymTD show that learning in faster

symbolic environments such as SILG can transfer to their visual counterparts, and allows certain perception challenges to be factored out.

Future work We find that some of the most challenging aspects of situated interactive language grounding include (1) grounding text references to entities without lexical overlap, (2) choosing from large textual action spaces, and (3) interpreting complex natural language descriptions. On the methodology front, further work is needed to investigate how to effectively use pretrained LMs for language grounding. Moreover, apart from recurrent state tracking, the other model enhancements do not yield significant gains on environments other than the ones they were proposed for. These results highlight the need for modelling techniques that generalize across environments.

SIR suggests that with additional improvements, it may be possible to have a performant model with the same architecture (but trained independently) across environments. Future work may explore whether (1) a single model with the same parameters can accomplish all tasks, (2) a single model with pretraining can be quickly finetuned on each task, and (3) learning in one environment is transferable to another. We believe SILG is well-suited to help answer these questions. Furthermore, SILGis designed to be easily extensible, with opportunities to add additional environments in the future.

3.5 Related Work

Benchmarks for NLP and RL. NLP benchmarks helped the development of models that generalize across different tasks [Wang et al., 2018, 2019a]. Similar benchmarks have furthered research in RL [Chevalier-Boisvert et al., 2018; Cobbe et al., 2020; Tassa et al., 2020]. SILG is the first benchmark for symbolic interactive language grounding with a diverse set of language and RL challenges. SILG evaluates generalization to new dynamics with (RTFM) and without lexical cues (Messenger) between text and entities, large partially observed worlds (SILGNetHack), large actions spaces (ALFWorld), and complex natural language instructions in rich visual scenes (SymTD). Finally, SILG provides a standard interface for symbolic interactive grounding environments via Gym, and considers the transfer to their visual counterparts (ALFWorld, SymTD). For reference, there is a host of perception-rich embodied environments not included in SILG due to the latter's emphasis on symbolic environments [MacMahon et al., 2006; Misra et al., 2017; Das et al., 2018; Anderson et al., 2018a; Ku et al., 2020a]. This emphasis allows SILG to provide an ef-

ficient benchmark for situated interactive language grounding. There are other complementary symbolic language grounding environments not included in this initial release of SILG due to time consideration such as [Chevalier-Boisvert et al., 2018; Ruis et al., 2020; Samvelyan et al., 2021]. We look forward to incorporating these in future iterations.

Interactive language grounding Language grounded policy-learning has been explored in the context of instruction following in tasks like navigation [Chen & Mooney, 2011; Hermann et al., 2017b; Fried et al., 2018a; Wang et al., 2019b; Daniele et al., 2017; Misra et al., 2017; Janner et al., 2018], games [Golland et al., 2010; Reckman et al., 2010; Andreas & Klein, 2015; Bahdanau et al., 2018; Küttler et al., 2020], and robotic control [Walter et al., 2013; Hemachandra et al., 2014; Blukis et al., 2019]. Touchdown, NetHack, and ALFWorld are three examples of such work included in SILG. While the above environments typically assume a small fixed set of world dynamics, other work explores settings where an agent must read text manuals to formulate appropriate policies for the game at hand. Branavan et al. [2012b] developed an agent to play Civilization more effectively by reading the game manual. Narasimhan et al. [2018a] and Zhong et al. [2020b] used text descriptions of game dynamics to learn policies that generalize to new environments and dynamics, without requiring feature engineering. Unlike these two works, Hanjie et al. [2021b] does not assume initial lexical overlap between entities in the world and entity references in the text manual. RTFM and Messenger are two examples of such work included in SILG.

Generalization to new environments in interactive language grounding In previous instruction following work, evaluation environments typically differ from training in their world observations. These difference range from differences in object placement in the same/new rooms (e.g. ALFWorld) to procedural generation of large game levels (e.g. NetHack). Moreover, some study generalization to new compositional instructions (e.g. Touchdown). Recent works explore generalization to new environment dynamics, which must be inferred by reading. These range from multi-step reasoning across texts (e.g. RTFM) to grounding entities to new text references (e.g. Messenger). The environments in SILG explore a variety of these generalization challenges. Many modelling techniques have been proposed to address these generalization challenges, including environmental variations [Hill et al., 2020a], memory structures [Hill et al., 2021], pretrained language models [Hill et al., 2020b], incremental guidance [Co-Reyes et al., 2019], subgoal-specification [Andreas et al., 2017], and hierarchical RL [Oh et al., 2017]. Our baseline and analy-

ses explores some of these techniques, including bidirectional feature-wise linear modulation [Zhong et al., 2020b], recurrent state-tracking [Küttler et al., 2020], entity-centric convolution [Küttler et al., 2020], entity-centric attention [Hanjie et al., 2021b], and pretrained language modelling [Hill et al., 2020b].

3.6 Using SILG

We use OpenAI Gym to create a common interface for all five SILG environments. For RTFM, Messenger, and ALFWorld, we create wrappers for the original environments such that the output of the Gym environment adheres to the shared interface. The custom environment variants we create for NetHack and for Touchdown are respective described in detail in Section 3.8 and 3.9.

To instantiate a SILG environment, the user needs to simply instantiate its Gym instance as follows.

```
from silg import envs
import gym
import random
env = gym.make('silg:td_segs_train-v0', time_penalty=-0.02)
obs = env.reset()
action = random.choice(list(range(len(env.action_space))))  # e.g. 0
obs, reward, done, info = env.step(action)
```

The `obs` dictionary then contains the environment outputs specified in Section 3.2.

3.7 Bidirectional Feature Wise Linear Modulation layer

Feature-wise linear modulation (FiLM), which modulates visual inputs using representations of text inputs, is an effective method for image captioning [Perez et al., 2018] and instruction following [Bahdanau et al., 2018]. Zhong et al. [2020b] extends FiLM to its bidirectional variant FiLM^2, which they show to be effective for modeling joint multi-hop references between visual and multiple text inputs. We find FiLM^2 to be an effective building block for the tasks considered in SILG.

Let + and * symbols denote element-wise addition and multiplication operations that broadcast over spatial dimensions. Let $\boldsymbol{x}_{\text{text}}$ denote a fixed-length $d\text{text}$-dimensional representation of the text and $\boldsymbol{X}_{\text{vis}}$ the

representation of visual inputs with height H, width W, and d_{vis} channels. Let Conv denote a convolution layer. FiLM^2 first modulates visual features using text features:

$$\boldsymbol{\gamma}_{\text{text}} = \boldsymbol{W}_{\gamma}\boldsymbol{x}_{\text{text}} + \boldsymbol{b}_{\gamma} \tag{3.8}$$

$$\boldsymbol{\beta}_{\text{text}} = \boldsymbol{W}_{\beta}\boldsymbol{x}_{\text{text}} + \boldsymbol{b}_{\beta} \tag{3.9}$$

$$\boldsymbol{V}_{\text{vis}} = \text{ReLU}((1 + \boldsymbol{\gamma}_{\text{text}}) * \text{Conv}_{\text{vis}}(\boldsymbol{X}_{\text{vis}}) + \boldsymbol{\beta}_{\text{text}}) \tag{3.10}$$

Then, it modulates text features using visual features:

$$\boldsymbol{\Gamma}_{\text{vis}} = \text{Conv}_{\gamma}(\boldsymbol{X}_{\text{vis}}) \tag{3.11}$$

$$\boldsymbol{B}_{\text{vis}} = \text{Conv}_{\beta}(\boldsymbol{X}_{\text{vis}}) \tag{3.12}$$

$$\boldsymbol{V}_{\text{text}} = \text{ReLU}((1 + \boldsymbol{\Gamma}_{\text{vis}}) * (\boldsymbol{W}_{\text{text}}\boldsymbol{x}_{\text{text}} + \boldsymbol{b}_{\text{text}}) + \boldsymbol{B}_{\text{vis}}) \tag{3.13}$$

The output of FiLM^2 is the sum of the modulated features $\boldsymbol{V}$ and its max-pooled summary $\boldsymbol{s}$ across spatial dimensions.

$$\boldsymbol{V} = \boldsymbol{V}_{\text{vis}} + \boldsymbol{V}_{\text{text}} \tag{3.14}$$

$$\boldsymbol{s} = \text{MaxPool}(\boldsymbol{V}) \tag{3.15}$$

3.8 Multitask NetHack

For NetHack, we create a multi-task environment that uniformly samples between the three tasks `Score`, `Gold`, and `Scout`. Given the sampled task, the agent observes a text string that specifies the goal (e.g. "get more gold"), in addition to the original environment text feedback to the agent's actions. For each task, we collect 10 human playthroughs where in a human plays the original NetHack Learning Environment and attempts to get the highest score possible within 50 steps. The empirical mean of these playthroughs is then used as the task's score threshold. In the SILG version of multi-task NetHack, the agent receives a reward of 1 if it exceeds the score threshold of the current task, and 0 otherwise. If the episode terminates without exceeding the score, then the agent receives -1. We find that this method of reward assignment strikes a

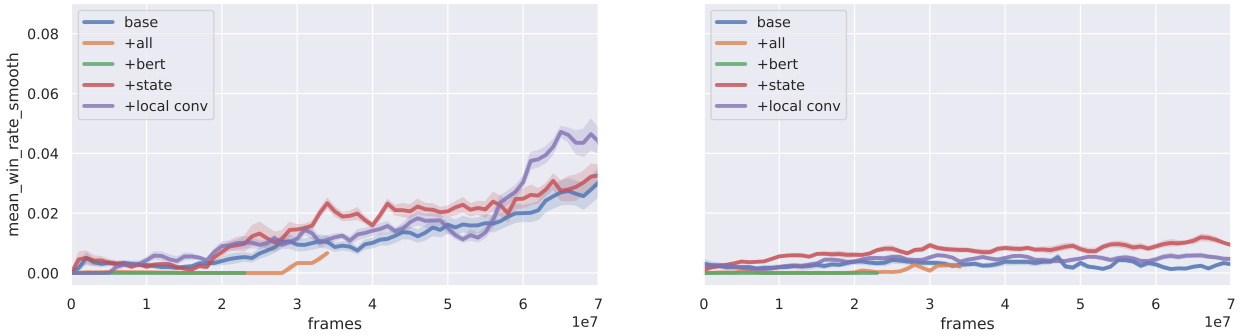


Figure 3.11: Manual SymTD performance. Left: train envs, right: validation envs.

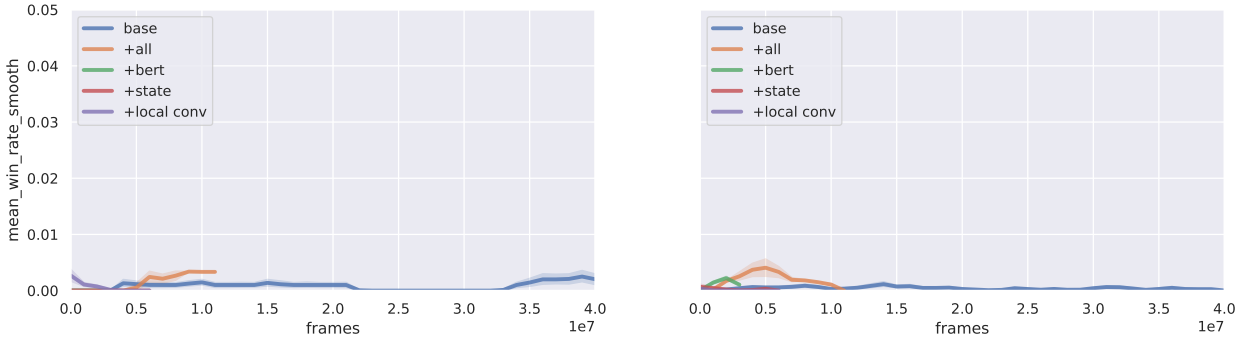


Figure 3.12: Manual VisTD performance. Left: train envs, right: validation envs.

balance between the very different reward distributions of the individual tasks (using the raw reward from individual tasks causes the agent to only learn to play `Scout`, the dominant task with frequent rewards). NetHack does not naturally provide train/validation/test splits. We create our own splits by splitting the seed ranges (1-1,000,000 for train, 1,000,001-2,000,000 for validation, 2,000,001-3,000,000 for test).

3.9 SymTD, VisTD, and Touchdown

Navigation In the original Touchdown implementation, the agent navigates with left, right, and forward commands. The left and right commands rotate the panorama at the current node so that the center of the panorama faces an adjacent node. The forward command then advances the agent to the node currently faced by the agent. We modify this navigation interface by fixing the panorama and providing the agent with a list of coordinates along the width-dimension of the panorama that corresponds to the locations of adjacent nodes that the agent may advance towards. The agent navigates by selecting one of the possible

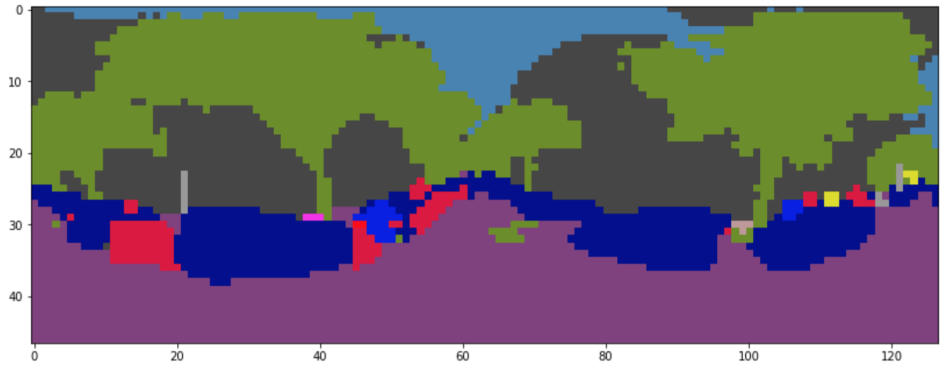


Figure 3.13: Segmentation map examples to create SymTD. The common objects the colours correspond to are sky (blue), buildings (gray), trees (green), sidewalk (pink), road (purple), cars (blue), traffic lights (yellow), and people (red). Note due to license agreements, this figure is a segmentation done on an example panorama provided directly on the StreetLearn website `https://sites.google.com/view/streetlearn/dataset`. Actual segmentations are visually very similar.

coordinates at each step. Our implementation allows the agent to see all possible navigation options upfront and reduces trajectory length by eliminating rotations. The setup is similar to Fried et al. [2018a] except our positional encoding embeds the distance of each point to the agent's current heading along the x-dimension, instead of using angle encodings.

Rewards Due to the sparsity of terminal ± 1 rewards, we provide a reward at each step by taking the difference in shortest path graph distance before and after the step (scaled by constant factor). This does not always assign positive reward when following the gold trajectory, but we find that it is a good heuristic in most cases.

SymTD We pass the raw panoramas from the original Touchdown task through a PSPNet [Zhao et al., 2017] trained on the Cityscapes dataset [Cordts et al., 2016]. The result is a segmentation map of the raw panorama with identical height and width dimensions. To allow for caching of the segmentation maps, we downsample the segmentation maps by taking a majority vote in each 23×23 patch. We found that the majority vote caused high-frequency classes (e.g. sky) to drown out low-frequency classes (e.g. pole). Therefore, we scale the vote of each class by its inverse count computed across all segmented panoramas. If $f(c)$ is the total count for class c, and P is a 23×23 patch in the segmentation map, the vote for class c in

patch P is:

$$v_P(c) = \frac{1}{f(c)^\alpha} \sum_{p \in P} \mathbb{1}[p = c]$$

The representative class for each patch P is then: $\max_{c \in C} v_P(c)$. We find that $\alpha = 1$ is effective at generating segmentation maps that preserve low-frequency classes. Figure 3.13 shows examples of such segmentation maps.

We conduct qualitative inspections of a sample of the segmented panoramas and observe that most segmentations are mostly correct relative to the input image. Despite this, human performance remains fairly low at approximately 60%. The main challenges faced by human players are (1) the symbolic features have no color information and (2) downsampling the segmentations result in highly pixelated figures, such that it is harder to distinguish smaller pedestrians from poles for example and (3) the navigation setup where the current heading is not necessarily the center of the panorama (indicated instead using an x-value) is extremely unintuitive for humans and often leads to the human player becoming disoriented. Given these observations, 60% may not be the upperbound for SymTD because controls unintuitive to humans do not affect ML models the same way.

VisTD We pass the raw panoramas through the ResNet-50 [He et al., 2016] backbone of a PSPNet trained on the Cityscapes dataset. We use the feature map from the last layer. Due to the large dimensionality along the feature axis and the difficulty caching these for efficient RL, we reduce the number of features using PCA to the top 10 principle components. The resulting feature maps for each panorama is $47 \times 128 \times 10$.

Manual stop TD In our variant of TD, the agent succeeds and the episode terminates immediately after the agent reaches the target node. We also include manual variants of SymTD and VisTD where the agent must manually select the "stop" option at the correct node. Thus, SymTD and VisTD are functionally equivalent to the original Touchdown environment.

The performance of our baseline as well as the baseline with various modelling advances are shown respectively in Figures 3.11 and 3.12 for Manual SymTD and Manual VisTD. Compared to SymTD and VisTD, the models largely fail to learn any reasonable policy within the allotted time. It remains an open

question whether the complex decision process associated with manual stopping Touchdown navigation is tractable using RL, without any supervised trajectories.

3.10 Collection of Human Expert Trajectories

The collection of human expert trajectories for purposes of establishing a performance upper bound is not very time-intensive. A player (paid 20$ per hour) who is familiar with text adventure games played through all five environments to collect the trajectories. Depending on the environment, the expert spent up to 30 minutes familiarizing themselves with the environment, then played approximately 50 episodes per environment, which are recorded to established human expert performance. During human playthroughs, the player is subject to the same step count limit as the RL agent. The maximum step count limit is 64 steps (for Touchdown), hence each episode is relatively quick in terms of play time.

For RTFM, Messenger, and NetHack, the human player observes a symbolic rendering of the grid along with a key that describes which symbol means which entity. The text is rendered below the grid. The human player then types in the command they would like to execute. For ALFWorld, the player observes the text rendering of the scene, as well as a list of text commands to choose from. The player then types in the index of the command they would like to execute. For Touchdown, the player observes a colour-coded rendering of the segmentation mask (of the panorama the player is in). x-coordinates are provided along the bottom of the segmentations, and a list of x-coordinates that the agent may advance towards at the next step is also provided. The player than chooses the index of the direction they would like to proceed in.

Playthrough interfaces for RTFM, Messenger, NetHack, ALFWorld, and SymTD are shown in Figure 3.14 through 3.18. Figure 3.13 shows examples of segmentation maps that the human player sees playing SymTD. Unfortunately we cannot include a figure of VisTD due to licensing agreement.

3.11 Licenses

We distribute SILG under a MIT LICENSE, which means that researchers are free to modify and distribute our software. The environments included in SILG use their own corresponding licenses. These are

1. RTFM: Attribution-NonCommercial 4.0 International

```
wall                wall                wall                wall                wall                wall

wall                _                   _                   shimmering spear    _                   wall

wall                you                 _                   _                   _                   wall

wall                _                   lightning wolf      fire panther        _                   wall

wall                _                   _                   gleaming morningstar_                   wall

wall                wall                wall                wall                wall                wall


JOINT TEXT
grandmasters beat cold . gleaming beat fire . shimmering beat lightning . blessed beat poison . jaguar are order of the
 forest . panther are rebel enclave . wolf are star alliance .
FIELD TEXT
task: defeat the rebel enclave
inv:

Reward: 0       Cumulative reward: 0    Steps: 0        Done: False     Your historical scores:
Type to choose action. Type ? to see action list.
```

Figure 3.14: Play interface for RTFM.

```
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           airplane    _           no_message  _           scientist   _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           robot       _           _           _           _
_           _           _           _           _           _           _           _           _           _
_           _           _           _           _           _           _           _           _           _

DYNAMICS TEXT
look to the researcher, this is crucial and your main task. that motionless robot is a dangerous foe. the plane that is not moving is the restr
icted message.
NON DYNAMICS TEXT
m1: that motionless robot is a dangerous foe.
m2: look to the researcher, this is crucial and your main task.
m3: the plane that is not moving is the restricted message.

Reward: 0       Cumulative reward: 0    Steps: 0        Done: False     Your historical scores:
Type to choose action. Type ? to see action list.
```

Figure 3.15: Play interface for Messenger.

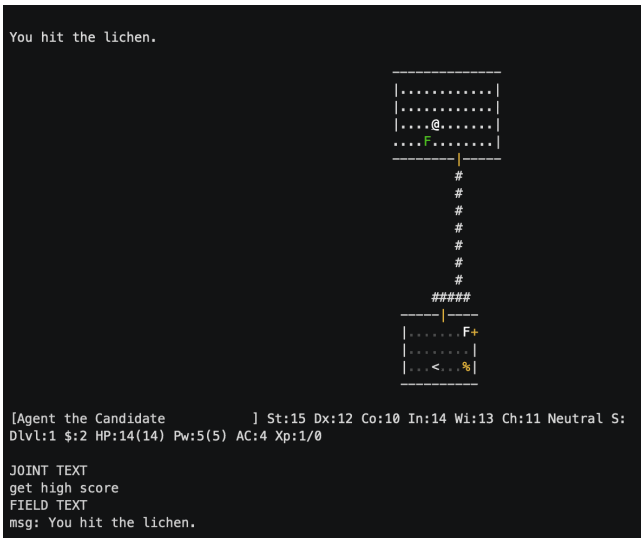


Figure 3.16: Play interface for NetHack.

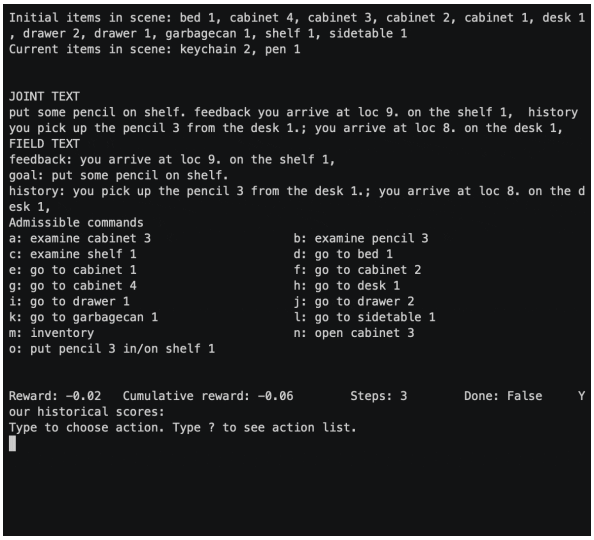


Figure 3.17: Play interface for ALFWorld.

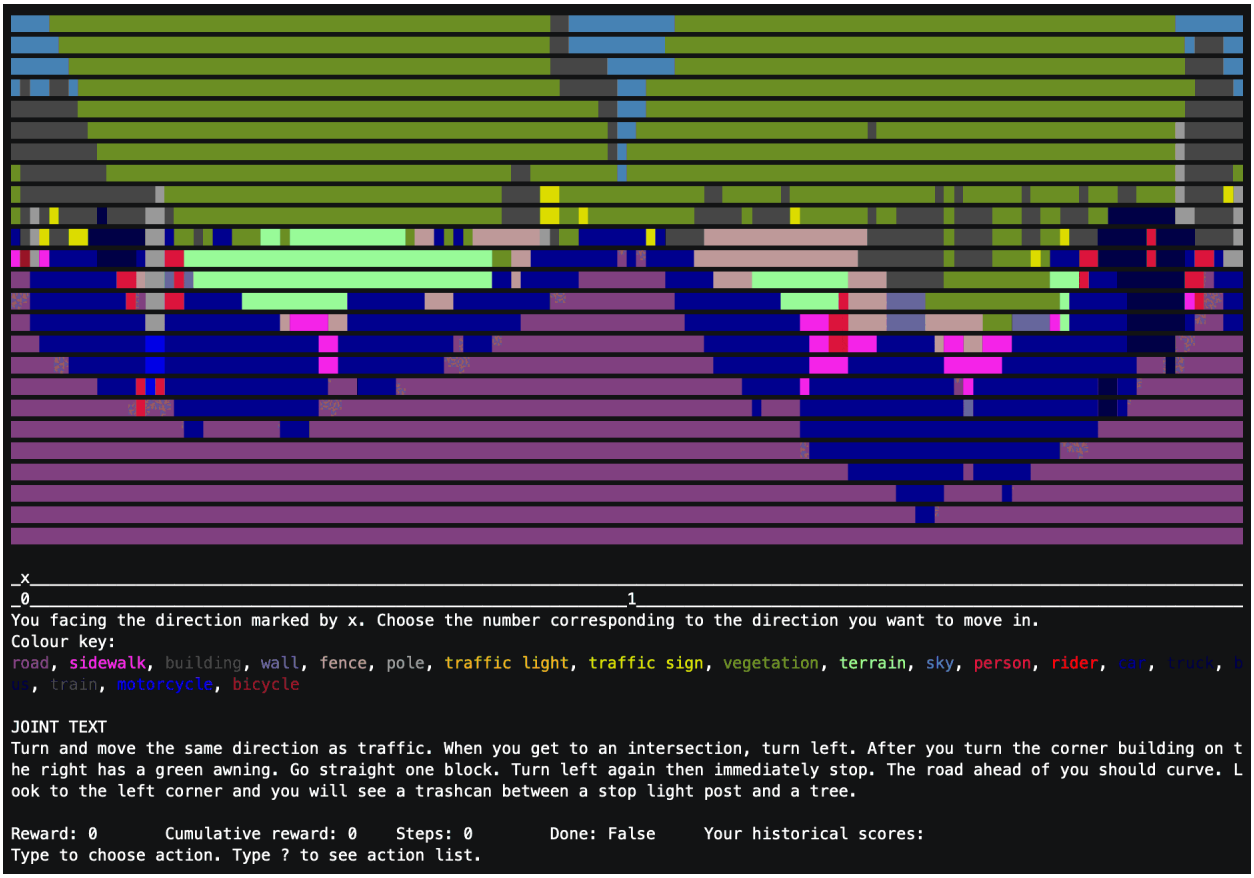


Figure 3.18: Play interface for VisTD.

2. Messenger: MIT

3. NetHack: NetHack General Public License

4. ALFWorld: MIT

5. Touchdown: Creative Commons Attribution 4.0 International

Of particular interest is Touchdown, whose raw panoramas come from Google Streetview. Neither we nor the creators of Touchdown distribute the panoramas. Users should follow instructions at `https://sites.google.com/view/streetlearn/dataset` to obtain the raw panoramas from Google.

3.12 Hyperparameters

By default, we use embedding size $d = 100$, and RNN size $r = 200$. The final representation H has size 400. We use 5 FiLM^2 layers. We train using Torchbeast [Küttler et al., 2019b] with an entropy cost of 0.05, baseline cost of 0.5, discount factor of 0.99, step penalty of -0.02, unroll length 80, and learning rate of 0.0005. We optimize using RMSProp with an epsilon of 0.01 and alpha 0.99. For Torchbeast parallelization, we use 30 actors, learner batch size of 24, and 4 learner threads. To account for long text sequences, we use the Huggingface PruneBERT model fine-tuned and distilled on SQuAD [Sanh et al., 2020].

Due to GPU memory constraints, we reduce the model size for some environments. For NetHack, we use 30 embedding size, 100 RNN size, 8 actors, 8 batch size, and 64 unroll length. For ALFWorld, we use 10 batch size. For the Touchdown variants, we use 30 embedding size, 100 RNN size, 200 final representation size, 8 actors, 3 batch size, 64 unroll length, and 3 FiLM^2 layers.

3.13 Compute resources

To produce our experiments, we ran 7 models each for RTFM, Messenger, and NetHack. Moreover, we ran 5 models for ALFWorld, SymTD, VisTD, Manual SymTD, and Manual VisTD. In total, this resulted in $7 \times 3 + 5 \times 5 = 46$ models. We used 5 seeds for each model, resulting in 230 runs. Each run required up to 20 CPUs (Intel Xeon) and 1 GPU (NVIDIA Quadro Pascal) for up to 2 weeks on an internal cluster. In

total, we used approximately $20 \times 24 \times 2 \times 230 = 220,800$ CPU hours and $24 \times 2 \times 230 = 11040$ GPU hours.

3.14 Conclusion

We introduced SILG, a new benchmark for evaluating language grounded agents across unique challenges posed by five symbolic interactive environments. Using SILG, we proposed the first shared architecture and analyzed recent methodological advancements in grounded language learning across on these environments. We showed that a shared architecture achieves comparable result to environment-specific methods, and that most advances do not result in significant gains on environments other than the one they were designed for. This highlights the need for modelling techniques that generalize across environments. Finally, the most models significantly trail human performance on SILG, which suggests ample room for future work. We hope that SILG will provide a unified platform for evaluating future methodological advances.

Chapter 4

Efficient R2L for grounded problems

So far we have examined how to test for generalization via reading. In Section 2, we designed generalization tests that require reading the manual to solve new problems. Then, in Section 3, we looked at building general R2L models, where we designed a general reading policy that applies to a broad range of LG challenges. One limitation of the work that we have shown so far is that in order for Reading to Learn models to generalize, they must be trained on a large variety of environments and manuals using a large number of episodes. For many grounded problems such as navigating in the real world or creating a language interface (Figure 4.1), we cannot build large labeled datasets. How can we find or create data for these grounded problems? In this section, we will look at pretraining a dynamics model to build a good prior for how to associate language manuals and world observations to learn policies for grounded environments.

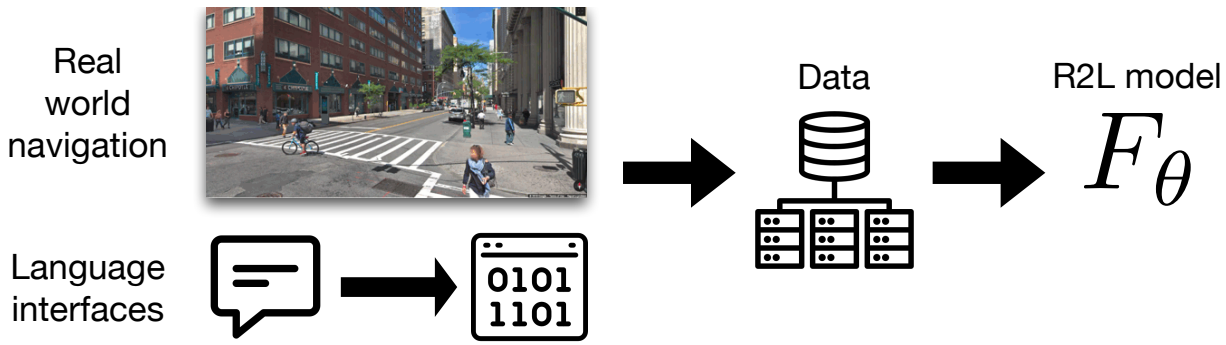


Figure 4.1: Finding and synthesizing data to train Reading to Learn efficiently for grounded problems

4.1 Introduction

Language is a powerful medium that humans use to reason about abstractions—its compositionality allows efficient descriptions that generalize across environments and tasks. Consider an agent that follows instructions to clean the house (e.g. find the dirty dishes and wash them). In tabula-rasa reinforcement learning (RL), the agent observes raw perceptual features of the environment, then grounds these visual features to language cues to learn how to behave through trial and error. In contrast, we can provide the agent with language descriptions that describe abstractions which are present in the environment (e.g. *there is a sink to your left and dishes on a table to your right*), thereby simplifying the grounding challenge. Language descriptions of observations occur naturally in many environments such as text prompts in graphical user interfaces [Liu et al., 2018], dialogue [He et al., 2018], and interactive games [Küttler et al., 2020]. Recent work has also shown improvements in visual manipulation [Shridhar et al., 2021] and navigation [Zhong et al., 2021; Tam et al., 2022] by captioning the observations with language descriptions. Despite these gains, learning how to interpret language descriptions is difficult through RL, especially on environments with complex language abstractions [Zhong et al., 2021].

We present Language Dynamics Distillation (`LDD`), a method that improves RL by learning a dynamics model on cheaply obtained unlabeled (i.e. no action labels) demonstrations with language descriptions. When learning how to use language descriptions effectively, one central challenge is how to disentangle language understanding from policy performance from sparse, delayed rewards. Our motivation is to learn initial language grounding via dynamics modeling from an offline dataset, away from the credit assignment and non-stationarity challenges posed by RL. While labeled demonstrations that tell the agent how to to act in each situation are expensive to collect, for many environments one can cheaply obtain unlabeled demonstrations (e.g. videos of experts performing the task) [Yang et al., 2019; Stadie et al., 2017]. Intuitively, `LDD` exploits these unlabeled demonstrations to learn how to associate language descriptions with abstractions in the environment. This knowledge is then used to bootstrap and more quickly learn policies that generalize to instructions and manuals in new environments. Given unlabeled demonstrations with language descriptions (e.g. captions of scene content), we first pretrain the model to predict the next observation given prior observations, similar to language modeling. A copy of this model is stored as a fixed teacher that grounds language descriptions to predict environment dynamics. We then train a model with RL, while

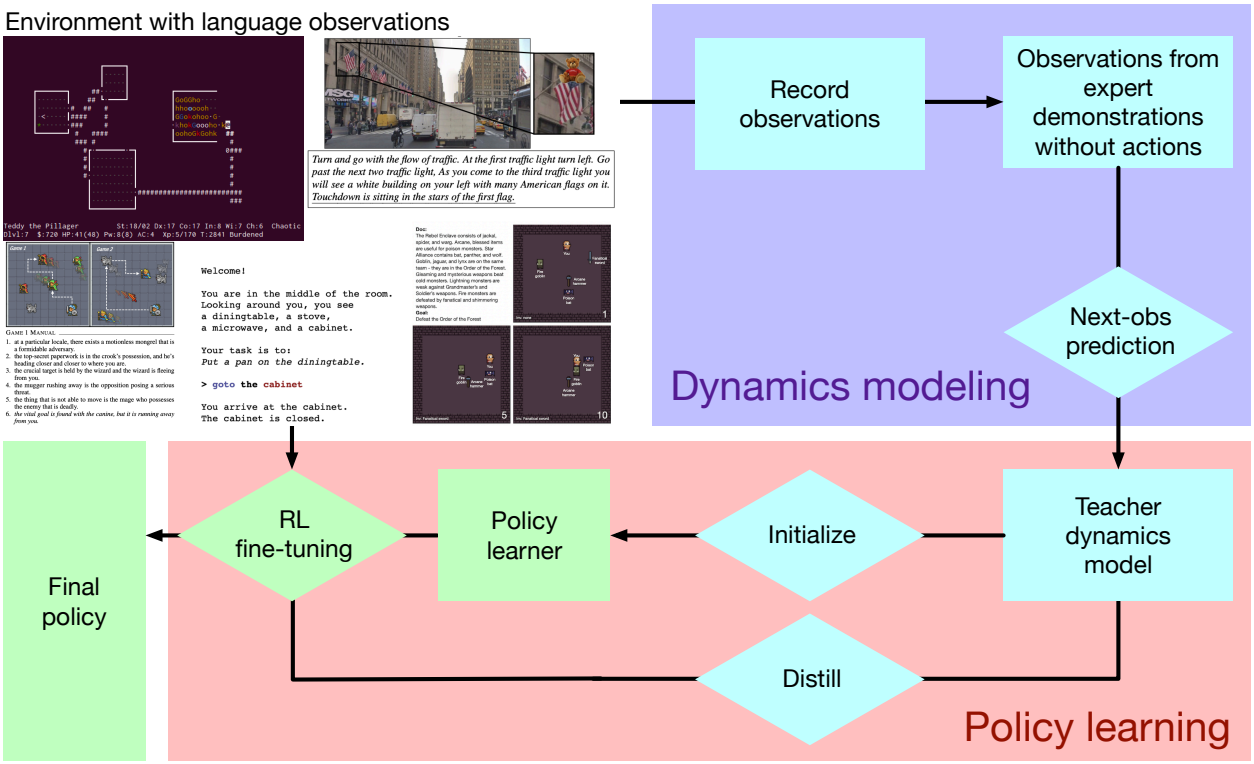


Figure 4.2: Language Dynamics Distillation (LDD). LDD uses cheap unlabeled demonstrations to learn a dynamics model of the environment, which is used to initialize and distill grounded representations into the policy learner. During the dynamics modeling phase (purple), we train a teacher model to predict the next observation given prior observations using unlabeled demonstrations. In the policy learning phase (red), we initialize a model with the teacher and distill intermediate representations from the teacher during reinforcement learning. The traditional policy learning loop is shown in green. LDD-specific components are shown in blue.

distilling intermediate representations from the teacher to avoid catastrophic forgetting of how to interpret language descriptions for dynamics modeling. In this way, the model learns to both maximize expected reward while retaining knowledge about how language descriptions relate to environment dynamics.

We evaluate LDD on the recent SILG benchmark [Zhong et al., 2021], which consists of five diverse environments with language descriptions including NetHack [Küttler et al., 2020], ALFWorld [Shridhar et al., 2021], RTFM [Zhong et al., 2020b], Messenger [Hanjie et al., 2021a], and Touchdown [Chen et al., 2018]. These environments present unique challenges in language-grounded policy-learning across complexity of instructions, visual observations, action space, reasoning procedure, and generalization. By learning a dynamics model from cheaply obtained unlabeled demonstrations, LDD consistently outperforms reinforcement learning with language descriptions both in terms of sample efficiency and generalization performance. Moreover, we compare LDD to other techniques that inject prior knowledge in VAE pretraining [Kingma & Welling, 2013], inverse reinforcement learning [Hanna & Stone, 2017; Torabi et al., 2018; Guo et al., 2019], and reward shaping with a pretrained expert [Merel et al., 2017]. LDD achieves top performance on all environments in terms of task completion and reward. In addition to comparing LDD to other methods, we ablate LDD to quantify the effect of language observations in dynamics modeling, and the importance of dynamics modeling with expert demonstrations. On two environments where we can control for the presence of language descriptions (NetHack game messages and Touchdown panorama captions), we show that language descriptions improve sample-efficiency and generalization. Finally, across all environments, we find that dynamics modeling with expert demonstrations is more effective than with non-expert rollouts.

4.2 Related Work

Learning by observing language. Recent work studies generalization to language instructions and manuals that specify new tasks and environments. These settings range from photorealistic/3D navigation [Anderson et al., 2018a; Chen et al., 2018; Ku et al., 2020b; Shridhar et al., 2020] to multi-hop reference games [Narasimhan et al., 2015; Zhong et al., 2020b; Hanjie et al., 2021a]. We use a collection of these tasks to evaluate LDD. There is also work where understanding language is not necessary to achieve the task, however its inclusion (e.g. via captions, scene descriptions) makes learning more efficient. Shridhar et al. [2021] show that one can quickly learn policies in a simulated kitchen environment described in text,

then transfer this policy to the 3D visual environment. Zhong et al. [2021] similarly transform photorealistic navigation to a symbolic form via image segmentation, then learn a policy that transfers to the original photorealistic setting. In work concurrent to ours, Tam et al. [2022] generate oracle captions of observations for simulated robotic control and city navigation, which improve policy learning. LDD is complementary to these—in addition to incorporating language descriptions as features, we show that learning a dynamics model from unlabeled demonstrations with language descriptions improves sample efficiency and results in better policies.

Imitation learning from observations. There is prior work on model-free as well as model-based imitation learning from observations. Model-free methods encourage the imitator to produce state distributions similar to those produced by the demonstrator, for example via generative adversarial learning [Merel et al., 2017] and reward shaping [Kimura et al., 2018]. In contrast, LDD only requires intermediate representations extracted from an expert dynamics model on states encountered by the learner, which are cheaper to compute than rollouts from an expert policy. Model-based approaches learn dynamics models that predict state-transitions given the current state and an action. Hanna & Stone [2017] learn an inverse model to map state-transitions to actions, which is then used to annotate unlabeled trajectories for imitation learning. Edwards et al. [2019] learn a forward dynamics model that predicts future states given state and latent action pairs. In contrast, LDD does not assume priors over the action space distribution. For instance, on ALFWorld, our method works even though it is impossible to enumerate the action space. In our experiments, we extend model-free reward shaping and model-based inverse dynamics modeling to account for language descriptions and compare LDD to these methods.

Representation learning in RL. In representation learning for RL, the agent learns representations of the environment using rewards and objectives based on the difference between the state and prior states [Strehl & Littman, 2008], raw visual observations [Jaderberg et al., 2017], learned agent representations [Raileanu & Rocktäschel, 2020], and random network observations [Burda et al., 2019]. In intrinsic exploration methods [Raileanu & Rocktäschel, 2020; Burda et al., 2019], the training objective encourages dissimilarity (e.g. in observation/state space) to prior agent experience so that the agent discovers novel states. Unlike intrinsic exploration, the distillation objective in Language Dynamics Distillation encourages similarity to

expert behaviour, as opposed to dissimilarity to prior agent experience. In reconstruction based representation learning methods [Strehl & Littman, 2008; Jaderberg et al., 2017], the training objective encourages the agent to learn intermediate representations that also capture the dynamics and structure of the environment by reconstructing the observations (e.g. predicting what objects are in scene). Language Dynamics Distillation is similar to reconstruction methods for representation learning, however unlike the latter, the dynamics model in LDD is trained on trajectories obtained from an expert policy as opposed to the agent policy. Language Dynamics Distillation is complementary to intrinsic exploration methods and to reconstruction based representation learning methods.

4.3 Language Dynamics Distillation

Recent work improves policy learning by augmenting environment observations with language descriptions [Shridhar et al., 2021; Zhong et al., 2021; Tam et al., 2022]. For environments with complex language abstractions, however, learning how to associate language to environment observations is difficult through RL due to sparse, delayed rewards. In Language Dynamics Distillation (LDD), we pretrain the model on unlabeled demonstrations (i.e. no annotated actions) with language descriptions to predict the dynamics of the environment, then fine-tune the language-aware model via RL. LDD consists of two phases. In the first dynamics modeling phase, we pretrain the model to predict future observations given unannotated demonstrations. We store a copy of the model as a fixed teacher that has learned grounded representations useful for predicting how the environment behaves under an expert policy. In the second reinforcement learning phase, we fine-tune the model through policy learning, while distilling representations from the teacher. This way, the model is trained to both maximize expected reward and retain knowledge about the dynamics of the environment. Fig 4.2 illustrates the components of LDD.

4.3.1 Background

Markov decision process. Consider a MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, r, \gamma\}$. Here, $\mathcal{S}$ and $\mathcal{A}$ respectively are the discrete state (e.g. language goals, descriptions, visual observations) and action spaces of the problem. $P(s_{t+1}|s_t, a_t)$ is the transition probability of transitioning into state s_{t+1} by taking action a_t from state s_t. $r(s, a)$ is the reward function given some state and action pair. γ is a discount factor to prioritize short-term

rewards.

Actor-critic methods for policy learning. In RL, we learn a policy $\pi(s;\theta)$ that maps from observations to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Let $\mathcal{R}(\tau)$ denote the total discounted reward over the trajectory τ. The objective is to maximize the expected reward $J_\pi(\theta) = \mathbb{E}_\pi[\mathcal{R}(\tau)]$ following the policy π by optimizing its parameters θ. For trajectory length T, the policy gradient is

$$\nabla \mathbb{E}_\pi[\mathcal{R}(\tau)] \quad = \quad \mathbb{E}_\pi\left[\left(\mathcal{R}(\tau)\sum_{t=1}^{T}\nabla \log \pi(a_t, s_t)\right)\right] = \mathbb{E}_\pi\left[\left(\sum_{t=1}^{T} G_t \nabla \log \pi(a_t, s_t)\right)\right] \tag{4.1}$$

where $G_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k+1}$ is the return or discounted future reward at time t. We consider the actor-critic family of policy gradient methods, where a critic is learned to reduce variance in the gradient estimate. Let $V(s) = \mathbb{E}_\pi[G_t|s_t = s]$ denote the state value, which corresponds to the expected returns by following the policy π from a state s. Actor critic methods estimate the state value function by learning another parametrized function V to bootstrap the estimation of the discounted return G_t. For instance, with one-step bootstrapping, we have $G_t \approx r_{t+1} + \gamma V(s_{t+1};\phi)$. The critic objective is then $J_V(\phi) = \frac{1}{2}\left(r_{t+1} + \gamma V(s_{t+1};\phi) - V(s_t;\phi)\right)^2$ We minimize a weighted sum of the policy objective and the critic objective $J_{\text{ac}}(\theta,\phi) = -J_\pi(\theta) + \alpha_V J_V(\phi)$.

4.3.2 Dynamics modeling during pretraining

In addition to policy learning, Language Dynamics Distillation learns a dynamics model from unlabeled demonstrations to initialize and distill into the policy learner. Consider a set of demonstrations without labeled actions $\mathcal{T}_\sigma = \{\tau_1, \tau_2, \ldots \tau_n\}$ obtained by rolling out some policy $\sigma(a_t, s_t)$, where each demonstration $\tau = [s_1, s_2, \ldots s_T]$ consists of a sequence of observations. We learn a dynamics model $\delta(s_1 \ldots s_t;\zeta)$ to predict the next observation s_{t+1} given the previous observations.

$$J_\delta(\zeta) \quad = \quad \frac{1}{nT}\left(\sum_{i=1}^{n}\left(\sum_{t=1}^{T} sim\left(s_{t+1}, \delta(s_1, \ldots s_t;\zeta)\right)\right)\right) \tag{4.2}$$

where sim is a differentiable similarity function between the predicted state $\delta(s_1, \ldots s_t)$ and the observed state s_{t+1}, and ζ are parameters of the dynamics model. In the environments we consider, sim is the cross-

entropy loss across a grid of symbols denoting entities present in the scene.

4.3.3 Dynamics distillation during policy learning

Fig 4.2 shows the decomposition of the model into a representation network f_{rep}, a policy head f_π, a value head f_V, and a dynamics head f_δ. The three heads share parameters because their inputs are formed by the same representation network.

$$\pi(s_t) = f_\pi\left(f_{\text{rep}}\left(s_t;\theta_{\text{rep}}\right);\theta_\pi\right) \tag{4.3}$$

$$V(s_t) = f_V\left(f_{\text{rep}}\left(s_t;\theta_{\text{rep}}\right);\theta_V\right) \tag{4.4}$$

$$\delta(s_t) = f_\delta\left(f_{\text{rep}}\left(s_t;\theta_{\text{rep}}\right);\theta_\delta\right) \tag{4.5}$$

In the first phase of dynamics modeling, we pretrain the model to predict future observations given demonstrations by optimizing J_δ. We then store a copy of the model as a fixed teacher $\tilde{\delta}(s_t) = \tilde{f}_\delta\left(\tilde{f_{\text{rep}}}\left(s_t;\tilde{\theta}_{\text{rep}}\right)\right)$. Let $|X-Y|$ denote the L2 distance between X and Y. During the second phase, in addition to policy learning, we optimize a distillation objective $J_d(\theta_{\text{rep}},\theta_\pi,\theta_V) = |\tilde{f_{\text{rep}}}\left(s_t;\tilde{\theta}_{\text{rep}}\right) - f_{\text{rep}}\left(s_t;\theta_{\text{rep}}\right)|$ to avoid catastrophic forgetting of how to interpret language descriptions for dynamics modeling. This quantity is the similarity between the feature representation produced by the fixed teacher (e.g. the pretrained dynamics model) and the feature representation produced by the model. Because $\tilde{\delta}$ is frozen, the parameters $\tilde{\theta}_{\text{rep}}$ are not included in the objective function J_d. The joint loss for Language Dynamics Distillation is then

$$J(\theta_{\text{rep}},\theta_\pi,\theta_V) = -J_\pi(\theta) + \alpha_V J_V(\phi) + \alpha_d J_d(\theta_{\text{rep}},\theta_\pi,\theta_V) \tag{4.6}$$

To summarize, using unlabeled demonstrations with language descriptions, `LDD` learns a dynamics model of the environment that grounds language descriptions to environment observations (Section 4.3.2). This prior knowledge is then injected into reinforcement learning via initialization and distillation (Section 4.3.3).

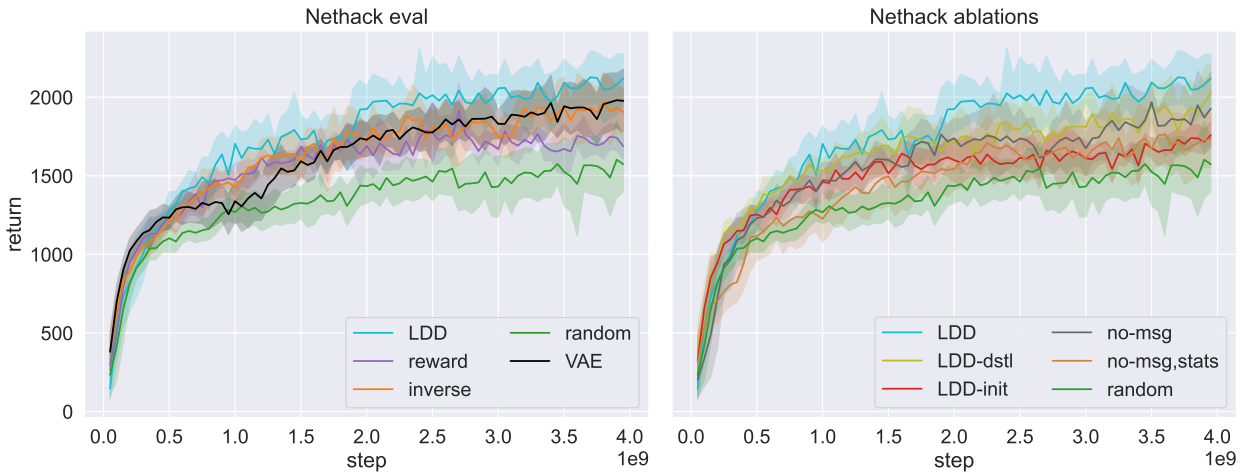


Figure 4.3: NetHack Challenge comparisons (left) and ablations (right). LDD consistently outperforms other methods.

4.4 Experiments

We evaluate Language Dynamics Distillation on the Situated Interactive Language Grounding benchmark (SILG) [Zhong et al., 2021]. SILG consists of five different language grounding environments with diverse challenges in term of complexity of observation space, action space, language, and reasoning procedure. In all environments, a situated agent observes symbolically (RTFM, Messenger, Nethack, Touchdown) or prose (ALFWorld) rendered visuals and interacts with the environment to follow some instance-specific language goals (e.g. what to do). In addition, the agent observes text manuals describing instance-agnostic environment rules (e.g. entity-role associations). The learning challenge is to learn a reading agent that generalizes to new environments with different environment rules (e.g. new entity-team associations, new parts of the map). The five different environments are as follows.

4.4.1 Environments

NetHack [Küttler et al., 2020]: The agent must descend a procedurally generated dungeon. Its primarily challenge is in large state space and partial observability, as the map remains obscured until exploration. Observations include a symbolic grid of entity IDs, in-game message, and description of character stats. The agent chooses among a fixed set of actions such as movement, picking up/buying/selling items, and attacking. We evaluate not on SILGNethack but the full NetHack challenge, a difficult game for humans

with $\sim$15% expert win rate [Street, 2013].

SymTouchdown [Zhong et al., 2021]: A symbolic version of Touchdown [Chen et al., 2018] where the agent navigates segmentation maps of Street View panoramas following long instructions. The primary challenge is reading long, natural instructions that describe photorealistic images. Evaluation is on new navigation instructions. Observations include a grid of segmentation class IDs corresponding to a discretized Google Street View panorama, synthetic captions of where objects are relative to the agent (e.g. *to your right you see a lot of road and some cars*), and language navigation instructions. The agent selects from a list of radial directions to proceed to the next panorama.

ALFWorld [Shridhar et al., 2021]: The agent navigates and manipulates objects inside a kitchen which is described via textual descriptions. ALFWorld is challenging due to its large (>50) text action space that vary across scenes. Evaluation is on unseen instructions. Observations include a textual description of the scene and language goals (e.g. *put a clean sponge on the metal rack*). The agent chooses among a variable set of language actions (e.g. *open drawer 1*).

RTFM [Zhong et al., 2020b]: The agent interprets a game manual and instruction to acquire the correct items to fight the correct monsters. Its main challenge is in multi-step reasoning that combines world observations with texts describing multiple entities. Evaluation is on a set of manuals distinct from those in training; hence the agent cannot memorize training manuals and must learn to read correctly in order to generalize. Observations include a symbolic grid containing names of entities present, manual describing high level game rules, agent inventory, and the language instruction. The agent chooses among a fixed set of movements. We train and evaluate on the first curriculum stage.

Messenger [Hanjie et al., 2021a]: The agent delivers a message from a source entity to a target while avoiding an enemy. The entities are referred to in text by many names, which have no lexical overlap with their symbol ID, hence the core challenge is in mapping language entity references in text to observed symbolic entity IDs. Evaluations are on new entity-role assignments (e.g. who carries the message). Observations include a symbolic grid containing symbol IDs of entities present, and a manual of entities and

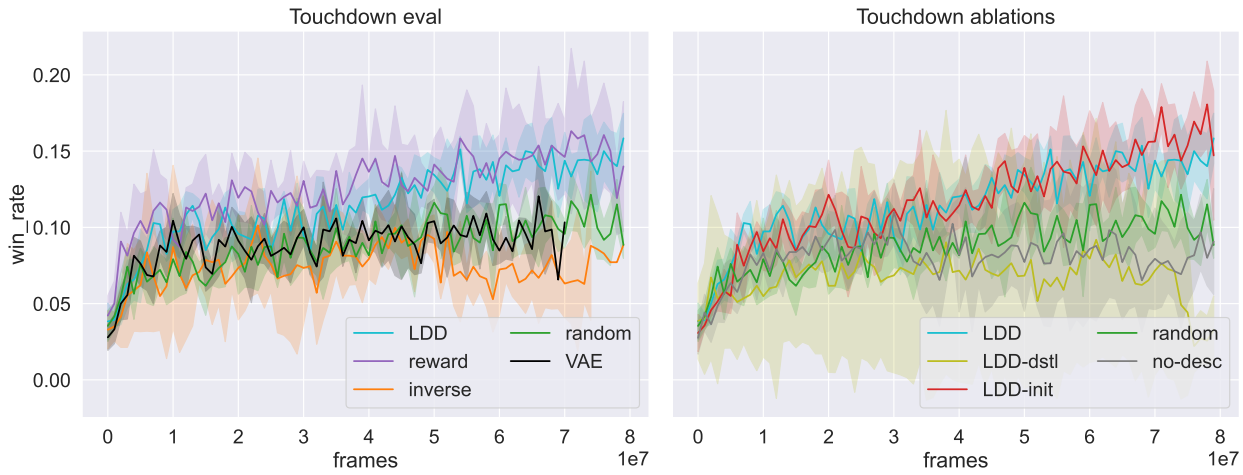


Figure 4.4: Touchdown comparisons (left) and ablations (right). Methods that distill or reward shape (`LDD`, `reward`, `LDD-init`) outperform those that do not.

roles. The agent chooses among a fixed set of movements. We train and evaluate on the second curriculum stage.

4.4.2 Method and Baselines

Reinforcement learning with language descriptions from scratch. We train a base tabula-rasa policy learner from random initialization. For NetHack, we train the base policy learner from [Küttler et al., 2020] using `moolib` [Mella et al., 2022]. For RTFM, ALFWorld, and Touchdown, we train the SIR model from [Zhong et al., 2021] using Torchbeast [Küttler et al., 2019a; Espeholt et al., 2018b]. For Messenger, we train the EMMA model from Hanjie et al. [2021a] using PPO [Schulman et al., 2017]. For NetHack, we train the base policy learner from [Küttler et al., 2020] using `moolib` [Mella et al., 2022]. For RTFM, ALFWorld, and Touchdown, we train the SIR model from [Zhong et al., 2021] using Torchbeast [Küttler et al., 2019a; Espeholt et al., 2018b]. For Messenger, we train the EMMA model from Hanjie et al. [2021a] using PPO [Schulman et al., 2017].

Pretraining representations via a variational autoencoder. We pretrain a variational autoencoder (VAE), a common approach for representation learning, that predicts the intermediate representation just before the policy head [Kingma & Welling, 2013]. This VAE has the same architecture as the policy learner, and is used to initialize the policy learner. The training procedure for the VAE is as described in Ha & Schmidhuber

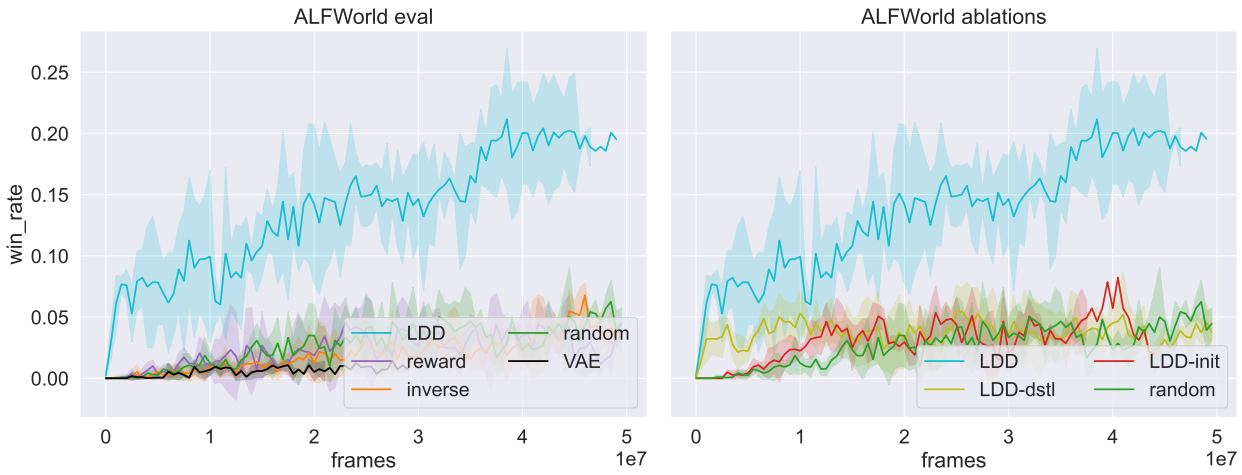


Figure 4.5: ALFWorld comparisons (left) and ablations (right). `LDD` consistently outperforms other methods.

[2018].

Language Dynamics Distillation (`LDD`) We train `LDD` variants of the baseline policy learners for each environment, where we pretrain the model to perform dynamics modeling on unannotated demonstrations. For NetHack, we use 100k screen-recordings (where actions are not annotated and cannot be trivially reverse engineered due to ambiguity in observations) of human-playthroughs from the alt.org NetHack public server. For Touchdown, we use unannotated demonstrations by human players. For ALFWorld, we use trajectories obtained from an A* planner with full state and goal knowledge, with actions removed. For RTFM and Messenger, we train expert trajectories until convergence, and sample 10k rollouts from the experts from which we remove action labels.

Reward shaping with expert. Methods such as Merel et al. [2017] reward shape with an expert by encouraging the agent to produce states similar to a demonstrator. To compare `LDD` to this idea, we use the dynamics model to predict the next observation under expert policy. The difference (e.g. accuracy in symbol prediction across grid) between the predicted observation and the actual observation after taking the action proposed by the agent is used as a penalty (e.g. negative auxiliary reward). This method is listing as `reward` in experiment figures and uses the same unlabeled demonstrations as `LDD`.

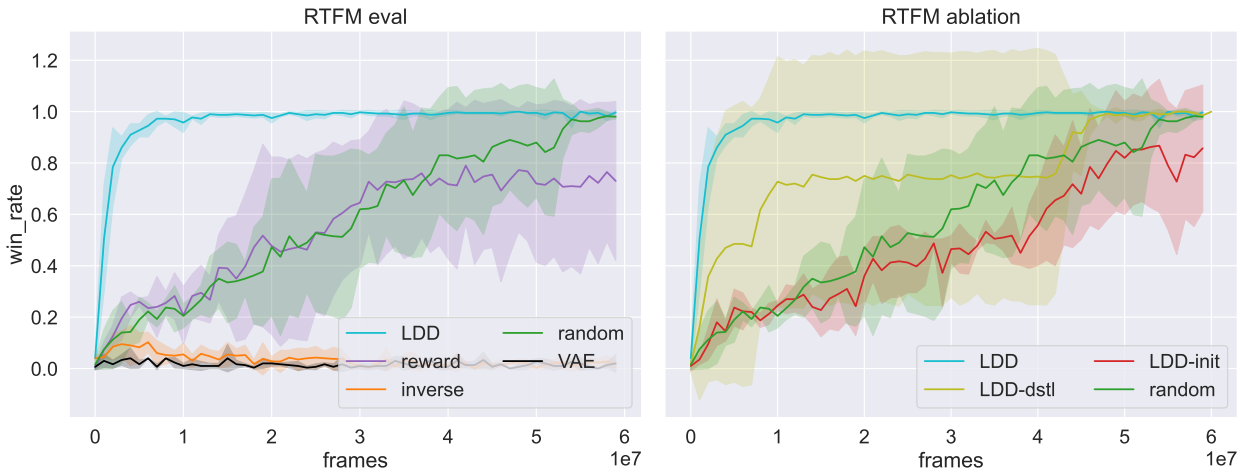


Figure 4.6: RTFM comparisons (left) and ablations (right). `LDD` consistently outperforms other methods. Because of the multi-step reasoning nature of RTFM solutions, partially complete strategies (e.g. able to do 2/3/4 cross-references) result in step-wise gains in win-rates. Strategies at different levels of completion result in larger variances when averaged.

Inverse reinforcement learning. Another class of methods learn a inverse dynamics model with which infer actions in unlabeled demonstrations, then learn to imitate the pseudo-labeled demonstrations [Hanna & Stone, 2017; Torabi et al., 2018]. To compare to this method, we first train the base policy learner for 10k episodes, then collect 10k rollouts to train an inverse dynamics model that predicts current action given current and future observations. This inverse model is used to annotate the original unlabeled demonstrations for imitation learning [Torabi et al., 2018]. Because these imitation policies do not generalize to novel environments and goals found in SILG evaluation, we additionally fine-tune them via RL similar to Guo et al. [2019]. This method is listing as `inverse` in experiment figures. In addition to the data used by `reward` and `LDD`, `inverse` uses additional rollouts to train the inverse dynamics model.

4.4.3 Results and ablations

`LDD` consistently improves performance across environments. We evaluate on held-out environments across 4 random seeds for NetHack in Fig 4.3, Touchdown in Fig 4.4, ALFWorld in Fig 4.5, RTFM in Fig 4.6, and Messenger in Fig 4.7. `LDD` obtains top performance compared to tabula-rasa policy learning with language descriptions, VAE pretraining, reward shaping using the dynamics model, and inverse reinforcement learning. This is consistent across challenges in multi-step reasoning (RTFM), language-entity

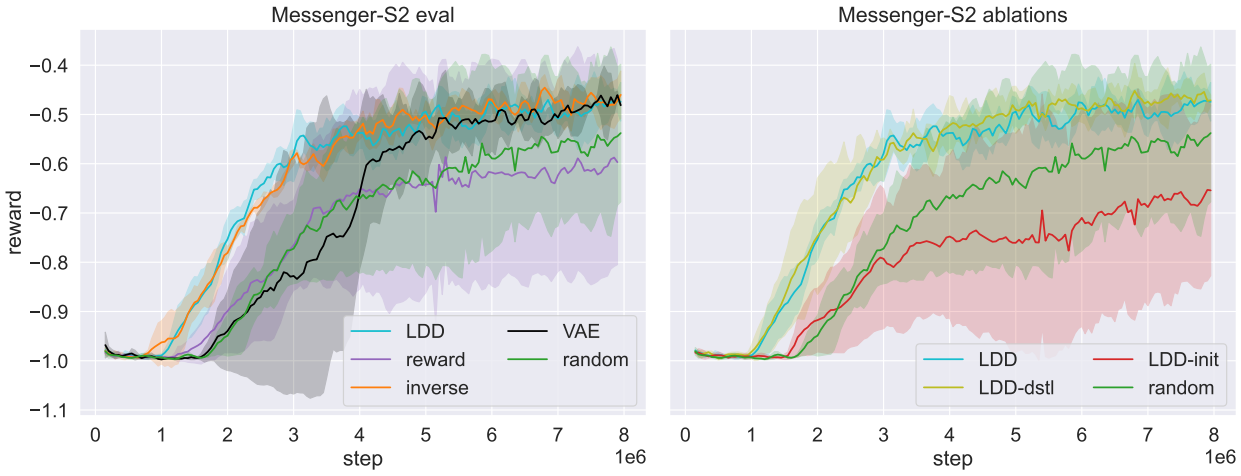


Figure 4.7: Messenger comparisons (left) and ablations (right). Methods that pretrain to initialize (`LDD`, `inverse`, `LDD-distill`) outperform those that do not.

generalization (Messenger), large language action spaces (ALFWorld), large procedurally generated states (NetHack), and long natural language instructions with complex visual scenes (Touchdown). We also ablate `LDD` by removing the initialization step (`LDD-init`) or the distillation step (`LDD-distill`). On Messenger, methods that pretrain to initialize (`LDD`, `inverse`, `LDD-distill`) outperform those that do not (`reward`, `LDD-init`). On Touchdown, methods that distill or reward shape (`LDD`, `reward`, `LDD-init`) outperform those that do not. Learning curves for each method across environments are shown in Section 4.11. `LDD` converges faster and to a higher win rate than other methods, with the exception of Touchdown, where it achieves lower training but higher evaluation win-rate.

Adding language descriptions improves performance. What is the role of language descriptions in pretraining and subsequent policy learning? To answer this question, we ablate language descriptions by removing them from the environments in NetHack and SymTouchdown (the other environments are not solvable without descriptions because they describe the objective). `no-msg` removes NetHack messages describing events near the agent (e.g. *kitten attacks the bat!*). `no-msg,stats` additionally removes character state descriptions (e.g. health, achievements, dungeon level). `no-desc` removes Touchdown captions that describe objects locations scene relative to the agent (e.g. *on your left, there are many building, some people, and few cars*). These variants differ only in the observation space used in dynamics modeling. In finetuning, they receive the same observation space with language descriptions. For both NetHack (Fig 4.3)

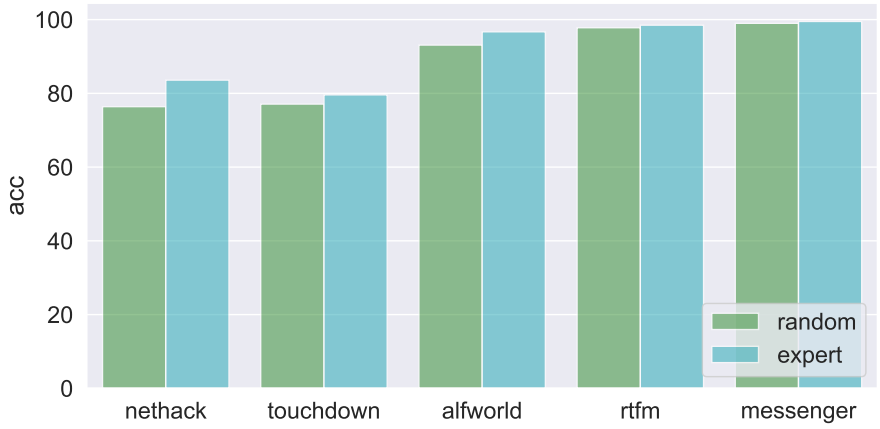


Figure 4.8: Dynamics model frame prediction accuracy using unlabeled expert demonstrations vs. rollouts from random policies. Training using expert demonstrations result in more accurate dynamics models, especially on complex environments that are difficult to explore via random policies.

and Touchdown (Fig 4.4), removal of in-game messages, character state descriptions, and synthetic captions degrade performance. Fig 4.13 further shows that adding language descriptions results in dynamics models with higher accuracy. This suggests that modeling language descriptions in the observation space result in better initialization and distillation, which improve subsequent policy learning.

Expert demonstrations cover late-stage strategy that result in asymptotic gains. In complex environments, experts demonstrate late stage strategies difficult to explore via random sampling. Because these stages are rarely reached, accurate dynamics models are especially helpful in providing signals when rewards are sparse. Take NetHack as an example: unlike experts, non-expert policies never proceed to the deeper dungeons of the game. A dynamics model trained on non-expert rollouts therefore struggles to generalize to unseen deeper dungeons. As the agent learns and descends deeper in the dungeon, a dynamics model trained on non-expert demonstrations results in less distillation gains. Fig 4.8 compares dynamics modeling from observations using expert demonstrations vs. using rollouts from a random policy. The evaluation is on a held-out set of expert demonstrations. Across environments, training on expert demonstrations outperforms training on non-expert demonstrations. This effect is less apparent on environments where random policies can (eventually) discover most of the state space (e.g, RTFM, Messenger) and more apparent on partially observed environments where only strategic expert policies can encounter rare states indicative of success (e.g. long-term planning in NetHack and Touchdown, choosing from large language

action space in ALFWorld).

4.5 Limitations

This work studies how language descriptions in unlabeled demonstrations benefit learning from observations. The environments used in this work are simulations. Despite variety across grounding challenges, performance on these environments do not necessarily transfer to other applications such as robotic control. A promising direction for future work is to investigate whether dynamics modelling on language observations show similar benefits in other applications.

4.6 Potential negative societal impacts

The methodology in this work are based on reinforcement learning, which may learn uninterpretable policies that achieve the objective in surprising ways (e.g. a robot that bumps along the cabinet while fetching dishes to clean). Language-conditioned policies are a way of controlling how policies behave by adjusting the language (e.g. instructions, in this case observations), however more research in this area is needed to develop methods that reliably understand and use language.

4.7 Code release

The source code for our experiments is available at

`https://github.com/vzhong/language-dynamics-distillation`.

4.8 Training details

We train and evaluate all methods on the SILG benchmark [Zhong et al., 2021], which comes with its own training and validation splits in terms of environment instances. We make distinction for Nethack, where we train and evaluate on the more difficult Nethack Challenge [Küttler et al., 2020], and Messenger, where we train and evaluate on the second curriculum stage as opposed to the first curriculum stage.

The code bases for each environments are based on the following work

1. Nethack: we use the https://github.com/facebookresearch/nle and its hyperparameters with the `human-monk` starting character. The demonstrations are 100k sampled `ttyrec` screen recordings downloaded from nethack.alt.org.

2. RTFM: we use the https://github.com/vzhong/silg and its default hyperparameters. The demonstrations are 10k sampled trajectories from a converged agent released with SILG.

3. Messenger: we use the https://github.com/ahjwang/messenger-emma and its default hyperparameters on the second curriculum stage. The demonstrations are 10k sampled trajectories from a converged agent trained using the default settings in EMMA on stage 1, then adapted to stage 2 via curriculum learning.

4. Touchdown: we use the https://github.com/vzhong/silg and its default hyperparameters. The demonstrations are the 6.5k human trajectories from the orinal Touchdown dataset [Chen et al., 2018].

5. ALFWorld: we use the https://github.com/vzhong/silg and its default hyperparameters. The demonstrations are the 21k full state planner trajectories from the original ALFRED dataset [Shridhar et al., 2020].

The dynamics models trained on these demonstrations are re-used for the reward-shaping method. To collect data for inverse dynamics modelling, we train a policy using the same hyperparameters for each environment for 10k episodes, then sample 10k episodes from the resulting policy. The sampled 10k episodes are used to learn a inverse dynamics model where two consecutive frames are used as the input and the inverse model predicts the action that took place between the frames. This inverse model is then used to predict actions on the demonstrations. The (pseudo-labeled) demonstrations are then used for imitation learning. The hyperparameters of the imitation learner is the same as those of the LDD experiments. This imitation learned model is then fined-tuned with RL.

Code for running the environment is anonymously submitted in the link in Section 4.7. Hyperparameters for our experiments are obtained from Zhong et al. [2021] for RTFM, ALFWorld, and Touchdown; Hanjie et al. [2021a] for Messenger, and Küttler et al. [2020] for NetHack. They are reproduced in Table 4.1 for convenience.

Name	RTFM	Messenger	NetHack	ALFWorld	Touchdown
Base model	SIR	EMMA	ChaoticDwarf	SIR	SIR
Embedding size	100	256	128	100	30
RNN size	200		128	200	100
Final repr size	400	256	128	400	200
Num FiLM^2 layers	5			5	3
Entropy cost	0.05	0.05	0.001	0.05	0.05
Baseline cost	0.5	0.5	0.25	0.5	0.5
Optimizer	RMSProp	Adam	Adam	RMSProp	RMSProp
Learning rate	5e-5	1e-4	1e-4	5e-4	5e-4
Optim epsilon	0.01	1e-6	1e-6	0.01	0.01
RMSProp alpha	0.99			0.99	0.99
Adam beta1		0.99	0.99		
Adam beta2		0.999	0.999		
Num actors	30	30	128	30	8
Learner batch size	24	24	128	10	3
Learner threads	4	4	4	4	4
Unroll length	80		64	80	64

Table 4.1: Hyperparameter settings. The base models SIR, EMMA, and ChaoticDwarf are respectively described in Zhong et al. [2021], Hanjie et al. [2021a], and Küttler et al. [2020].

4.9 Compute resources

We use a slurm cluster to train models. Each machine is equipped with a NVIDIA GPU with at least 16GB RAM and 20 CPU cores. Each run typically last 3 days, with the exception of ALFWorld (10 days) and Touchdown (6 days). Across 5 environments, we run 4 methods and 2 ablations for a total of 6 experiments. We additionally run 2 more language ablations experiments for Nethack and 1 more for Touchdown. Each experiment consists of 4 random seeds for a total of $(5 \times 6 + 3) \times 4 = 132$ runs. For the policy learning stage, our resource usage are on the order of $132 \times 10 \times 24 = 32k$ GPU hours or $132 \times 10 \times 24 \times 20 = 634k$ CPU hours. These experiments compose the bulk of our resource usage.

For dynamics pretraining, each run takes approximately 2 days of 1 GPU and 4 CPU. We re-use the trained dynamics model for reward shaping experiments. Inverse-dynamics modelling additionally require 1 day of pretraining an initial non-expert policy, generating rollouts from said policy, learning a inverse-dynamics model, and annotating unlabeled demonstrations with the inverse-dynamics model.

4.10 Asset and license

We distribute this work under the MIT license. The dataset we use are publically available and distributed as a part of the SILG benchmark [Zhong et al., 2021]. There are no personally identifying information in the assets we use. SILG is distributed under a MIT license. The included environments are licensed as follows:

1. NetHack: NetHack General Public License
2. Touchdown: Creative Commons Attribution 4.0 International
3. ALFWorld: MIT
4. RTFM: Attribution-NonCommercial 4.0 International
5. Messenger: MIT

4.11 Learning curves

Note that NetHack does not have a held-out evaluation set of environments.

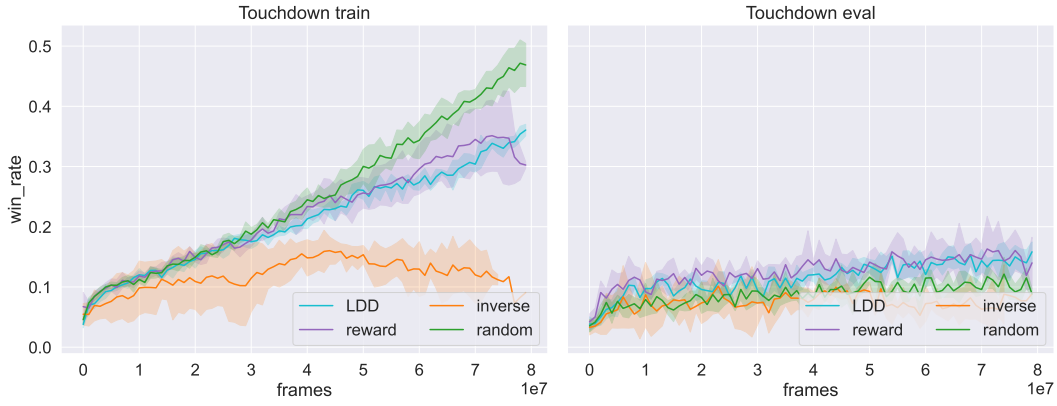


Figure 4.9: Touchdown learning curves

4.12 Conclusion

While recent work showed that augmentation with language descriptions result in better policies, learning how to ground language descriptions to observations is difficult through naive RL with sparse, delayed

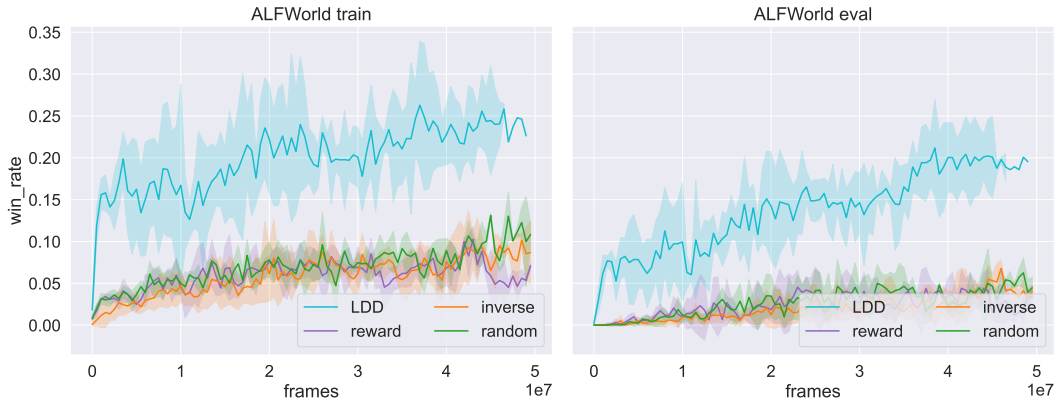


Figure 4.10: ALFWorld learning curves

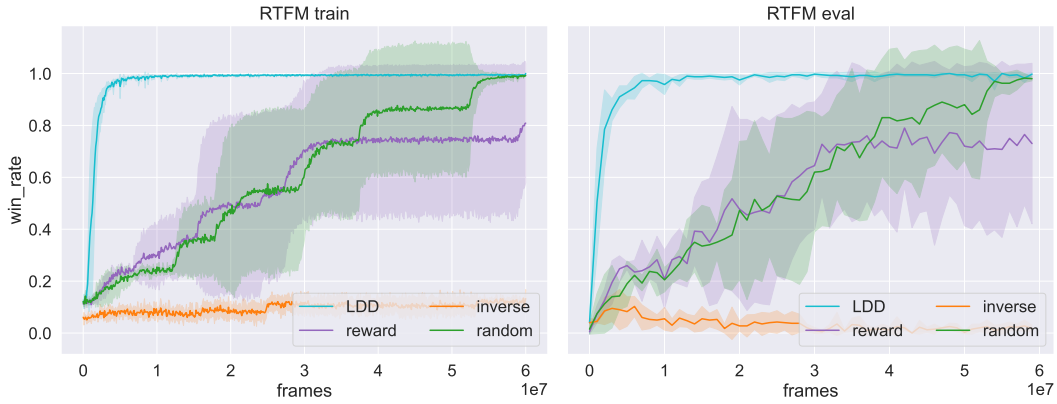


Figure 4.11: RTFM learning curves

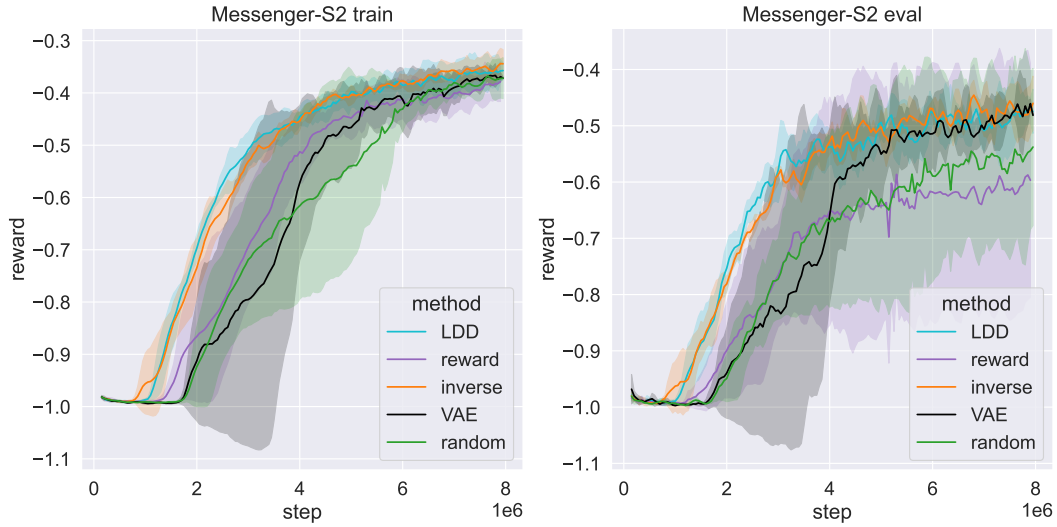


Figure 4.12: Messenger learning curves

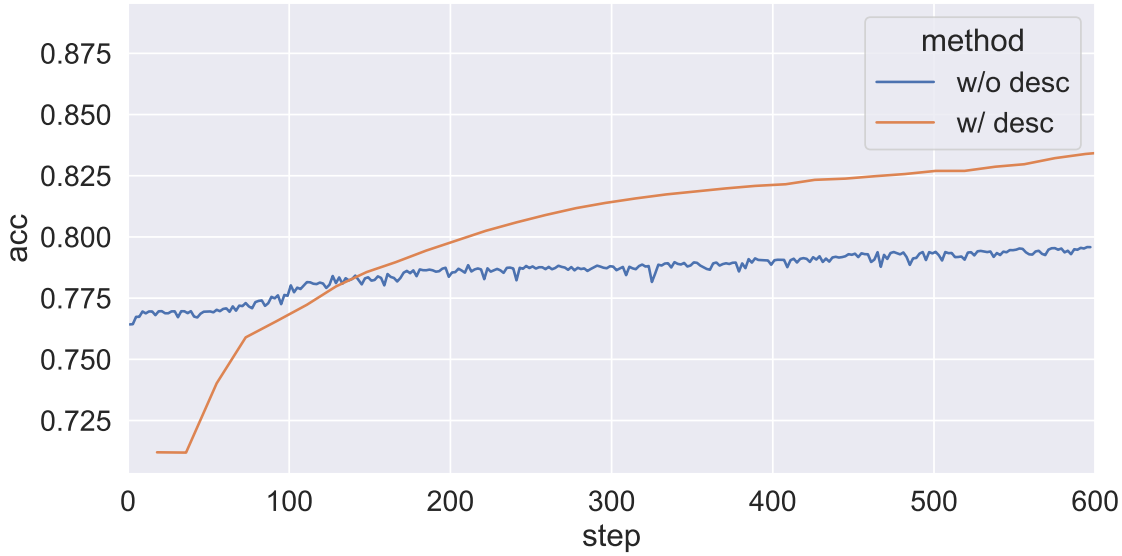


Figure 4.13: Dynamics modelling pixel-wise accuracy for SymTouchdown with vs. without language description inputs.

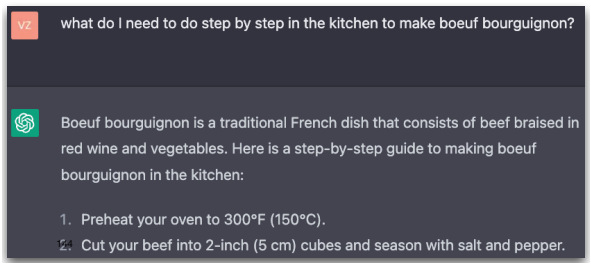
rewards. We proposed Language Dynamics Distillation, which pretrains a dynamics model using cheaply-obtained unlabeled demonstrations with language descriptions to initialize and distill into the policy learner. On five tasks with language descriptions, `LDD` improved sample efficiency and resulted in better policies than RL from scratch, inverse RL, and expert reward shaping. In addition, the benefit from initialization and distillation differ on an environment basis, but are complementary across environments. Moreover, language descriptions improved initialization and distillation gains in policy learning. Finally, learning to model dynamics with expert demonstrations was more effective than with non-expert rollouts. A promising direction for future research is studying whether dynamics modeling with language descriptions is similarly effective in robotic control where naive RL can be prohibitively expensive, but unlabeled demonstrations with synthetic captions are cheap to obtain.

Chapter 5

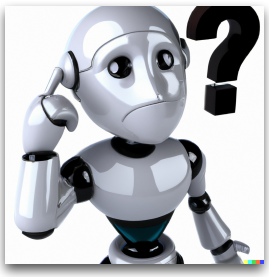
Discussion and future work

In this section, we will discuss future directions in Reading to Learn.

How can we Read to Learn in context? Many real-world applications require specific contextual knowledge. For instance, a cooking robot for the home cannot expect to train on a generic kitchen and then transfer to real-life kitchens in users' homes. However, large-scale pretraining is typically limited to general knowledge on the internet. Consider, for instance, asking ChatGPT the question "how do I cook Boeuf Bourginion?" (Figure 5.1, left). Because this model has been trained on the generic text on the internet, it gives a generic recipe of cooking instructions. How can we go from this generic recipe to an actionable plan, say, by our cooking robot in our kitchen? One answer to this question is to add contextual information. What this model is missing are things like what ingredients do we have? How do we prepare these ingredients? What appliances do we have? How do we operated these appliances?



(a) Asking ChatGPT "how do I cook boeuf bourginion"

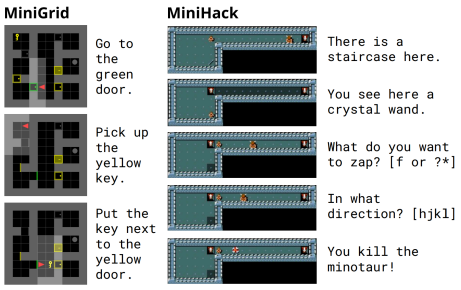


(b) Actively inquiring about grounded context.

Figure 5.1: How can we Read to Learn in context?

When a vehicle is detected running ahead of you, the system automatically decelerates your vehicle… The system will respond to changes in the speed of the vehicle ahead in order to maintain the vehicle-to-vehicle distance set by the driver… After the vehicle ahead starts off, pressing the "+RES" switch or depressing the accelerator pedal (start-off operation) will resume follow-up cruising…

(a) Subproblems for mastery of the cruise control system, as listed in a manual.



(b) Automatically extracting subproblems from language goals in [Mu et al., 2022].

(c) Synthesizing environments from a manual. Example from Yu et al. [2023].

Figure 5.2: How can we design currilua automatically by reading?

The first thread of future work has to do with automatically adapting general knowledge to specific grounded context. We examined one example of this in adapting pretrained models to specific grounded environments through dynamics modeling [Zhong et al., 2022]. Some future directions in this area include actively inquiring about the specific context through language. For instance, our cooking robot may ask the user "what do you use to drain liquid?", "where can I find this tool?", "how do I use this tool?". And using the answers the user gives to build up its own manual to inform its own downstream policy (Figure 5.1, right).

How can we design learning curricula automatically by reading? Curriculum learning is an effective way to solve complex problems by solving simpler subproblems. However, it is generally difficult to design a curriculum. For instance, consider learning a controller for the cruise control system of a car. What subskills are required to master the cruise control system? What are some scenarios that the controller must learn? More generally, given a complex problem, how do we decompose it into useful subproblems?

One way humans decompose problems into useful subproblems is by reading about the original problem. For instance, we can read the section about the cruise control system in the car manual to master the system

(Figure 5.2, top left). As we read this manual, we can start to imagine different relevant scenarios. For instance, there is a car that is driving ahead of our car and stopping, we want to resume cruising from a stop, and so on. What we are learning by reading the manual is what subproblems for an effective curriculum, in this case towards the mastery of the cruise control system.

The second thread of future work is on given a new problem, reading about the new problem to construct relevant subproblems, then forming a curriculum to learn more complex problems. One example of this is Mu et al. [2022] (Figure 5.2, top right), which repurposes language feedback in the environment as subproblems, then proposes these subproblems in a curriculum to learn more sample efficiently. Some future directions include environment synthesis by reading manuals. For instance, imagine we want to deploy the cooking robot to our home kitchen. As an user, we do not want the robot to RL in my kitchen, but perhaps we wouldn't mind describing to the robot my kitchen. With environment synthesis, we can read this description to generate a simulation of our kitchen [Yu et al., 2023], then adapt the robot's policy to our kitchen.

The promise of Reading to Learn is that instead of annotating data at scale, we can write language specifications that models read to generalize. In previous sections, we discussed the benefits of Reading to Learn over traditional Learning to Read, particularly with regards to generalization to new problems using language manuals as opposed to large labeled datasets. We then discussed three challenges in Reading to Learn. First, we developed a testbed to evaluate generalization via reading in RTFM [Zhong et al., 2020b]. Next, we developed SILG [Zhong et al., 2021], a Reading to Learn benchmark that evaluates a variety of language grounding challenges in multiple interactive environments, as well as a general reading policy that addresses these distinct grounding challenges. Finally, we explored ways to facilitate Reading to Learn through Language Dynamics Distillation [Zhong et al., 2022], where we pretrain the policy to associate language manuals to world observations by dynamics modelling on unlabelled demonstrations. The goal of Reading to Learn is to make machine learning accessible. There is perhaps no better time to work on Reading to Learn. One the one hand, distributed learning is becoming more and more practical due to advances in hardware and systems. On the other hand, large-scale language modelling allows us to build reading systems more capable than we have ever had previously. It is time to explore the role of language not as an end, but as a conduit for learning.

Bibliography

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018a.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018b.

Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. In *EMNLP*, 2015.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *ICML*, 2017.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *NAACL*, 2018.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to follow language instructions with adversarial reward induction. *arXiv preprint arXiv:1806.01946*, 2018.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to follow language instructions with adversarial reward induction. In *ICLR*, 2019.

K. Barnard and D. Forsyth. Learning the semantics of words and pictures. In *ICCV*, 2001.

Valts Blukis, Yannick Terme, Eyvind Niklasson, Ross A Knepper, and Yoav Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *CoRL*, 2019.

S. R. K. Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. In *ACL*, 2011.

S. R. K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *ACL*, 2012a.

S.R.K. Branavan. *Grounding Linguistic Analysis in Control Applications*. PhD thesis, MIT, 2012.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR*, 2019.

David L. Chen and Raymond J. Mooney. Learning to sportscast: A test of grounded language acquisition. In *ICML*, 2008.

David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. *San Francisco, CA*, pp. 859–865, 2011.

Howard Chen, Alane Suhr, Dipendra Kumar Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, 2018.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *ICLR*, 2019.

John D. Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. In *ICLR*, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, 2020.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Andrea F Daniele, Mohit Bansal, and Matthew R Walter. Navigational instruction generation as inverse reinforcement learning with neural machine translation. In *HRI*. IEEE, 2017.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *CVPR*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. In *ICML*, 2018.

Ashley D Edwards, Himanshu Sahni, Yannick Schroecker, and Charles L Isbell. Imitating Latent Policies from Observation. In *ICML*, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *ICML*, 2018a.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018b.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron,

Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018c.

Francis Ferraro, Nasrin Mostafazadeh, Ting-Hao Huang, Lucy Vanderwende, Jacob Devlin, Michel Galley, and Margaret Mitchell. A survey of current datasets for vision and language research. In *EMNLP*, 2015.

Daniel Fried, Jacob Andreas, and Dan Klein. Unified pragmatic models for generating and following instructions. In *NAACL*, 2018a.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *NeurIPS*, 2018b.

Dave Golland, Percy Liang, and Dan Klein. A game-theoretic approach to generating spatial descriptions. In *EMNLP*, 2010.

Xiaoxiao Guo, Shiyu Chang, Mo Yu, Gerald Tesauro, and Murray Campbell. Hybrid reinforcement learning with expert state sequences. In *AAAI*, 2019.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

Austin W. Hanjie, Victor Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *ICML*, 2021a.

Austin W. Hanjie, Victor Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *ICML*, 2021b.

Josiah P. Hanna and Peter Stone. Grounded Action Transformation for Robot Learning in Simulation. In *AAAI*, 2017.

He He, Derek Chen, Anusha Balakrishnan, and Percy Liang. Decoupling strategy and generation in negotiation dialogues. In *EMNLP*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

Sachithra Hemachandra, Matthew R Walter, Stefanie Tellex, and Seth Teller. Learning spatial-semantic representations from natural language descriptions and scene classifications. In *ICRA*, 2014.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3d world. *CoRR*, abs/1706.06551, 2017a.

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

Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. Environmental drivers of systematicity and generalization in a situated agent. In *ICLR*, 2020a.

Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020b.

Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. In *ICLR*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Compututation*, 9(8), 1997.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. *CoRR*, abs/1906.00744, 2019.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

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

Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *CoRR*, abs/1906.07343, 2019.

Daiki Kimura, Subhajit Chaudhury, Ryuki Tachibana, and Sakyasingha Dasgupta. Internal Model from Observations for Reward Shaping. In *ALA workshop at ICML*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv*, 1312.6114, 2013.

Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *HRI*, 2010.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2020a.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *EMNLP*, 2020b.

Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv*, 1910.03552, 2019a.

Heinrich Küttler, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette. TorchBeast: A PyTorch Platform for Distributed RL. *arXiv preprint arXiv:1910.03552*, 2019b. URL `https://github.com/facebookresearch/torchbeast`.

Heinrich Küttler, Nantas Nardelli, Alexander H Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack learning environment. In *NeurIPS*, 2020.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. In *EMNLP*, 2017.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *EMNLP*, 2016.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *ICLR*, 2018.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A Survey of Reinforcement Learning Informed by Natural Language. In *IJCAI*, 2019.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *ACL*, 2015.

Matt MacMahon, Brian J. Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, 2006.

Harsh Mehta, Yoav Artzi, Jason Baldridge, Eugene Ie, and Piotr Mirowski. Retouchdown: Adding touchdown to streetlearn as a shareable resource for language grounding tasks in street view. *arXiv preprint arXiv:2001.03671*, 2020.

Vegard Mella, Eric Hambro, Danielle Rothermel, and Heinrich Küttler. moolib: A platform for distributed RL. https://github.com/facebookresearch/moolib, 2022.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv*, 1707.02201, 2017.

Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, Denis Teplyashin, Karl Moritz Hermann, Mateusz Malinowski, Matthew Koichi Grimes, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, et al. The streetlearn environment and dataset. *NeurIPS*, 2018.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette. Improving intrinsic exploration with language abstractions. In *NeurIPS*, 2022.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, 2015.

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

Karthik Narasimhan, Regina Barzilay, and Tommi S. Jaakkola. Deep transfer in reinforcement learning by language grounding. *JAIR*, 2018b.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

OpenAI. Introducing chatgpt, 2023. URL `https://openai.com/blog/chatgpt`.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018.

Roberta Raileanu and Tim Rocktäschel. RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments. In *ICLR*, 2020.

Hilke Reckman, Jeff Orkin, and Deb Roy. Learning meanings of words and constructions, grounded in a virtual game. *Semantic Approaches in Natural Language Processing*, pp. 67, 2010.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Laura Ruis, Jacob Andreas, Marco Baroni, Diane Bouchacourt, and Brenden M. Lake. A benchmark for systematic generalization in grounded language understanding. In *NeurIPS*, 2020.

Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *NeurIPS Datasets and Benchmarks Track*, 2021.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*, 2020.

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

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *CVPR*, 2020.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *ICLR*, 2021.

Bradly C. Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv*, 1703.01703, 2017.

Chris Street. Observations on expert play in nethack. `https://codehappy.net/nethack/data.htm`, 2013.

Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *JCSS*, 2008.

Allison C. Tam, Neil C. Rabinowitz, Andrew K. Lampinen, Nicholas A. Roy, Stephanie C. Y. Chan, DJ Strouse, Jane X. Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations, 2022.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control. *arXiv preprint arXiv:2006.12983*, 2020.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.

T. Tieleman and G. Hinton. Lecture 6.5—RmsPropG: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

Matthew R Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. Learning semantic maps from natural language descriptions. In *RSS*, 2013.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP workshop at EMNLP*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, 2019a.

Sida I. Wang, Percy Liang, and Christopher D. Manning. Learning language games through interaction. In *ACL*, 2016.

Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *CVPR*, 2019b.

Jiannan Xiang, Xin Eric Wang, and William Yang Wang. Learning to stop: A simple yet effective approach to urban vision-language navigation. In *Findings of EMNLP*, 2020.

Chao Yang, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Huaping Liu, Junzhou Huang, and Chuang Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. In *NeurIPS*, 2019.

Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Dee M, Jodilyn Peralta, Brian Ichter, Karol Hausman, and Fei Xia. Scaling robot learning with semantically imagined experience, 2023.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

Victor Zhong and Luke Zettlemoyer. E3: Entailment-driven extracting and editing for conversational machine reading. In *ACL*, 2019.

Victor Zhong, Caiming Xiong, and Richard Socher. Global-locally self-attentive dialogue state tracker. In *ACL*, 2018.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. Grounded adaptation for zero-shot executable semantic parsing. In *EMNLP*, 2020a.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. RTFM: Generalising to Novel Environment Dynamics via Reading. In *ICLR*, 2020b.

Victor Zhong, Austin W Hanjie, Sida I Wang, Karthik Narasimhan, and Luke Zettlemoyer. SILG: The multi-environment symbolic interactive language grounding benchmark. In *NeurIPS*, 2021.

Victor Zhong, Jesse Mu, Luke Zettlemoyer, Edward Grefenstette, and Tim Rocktäschel. Improving policy learning via language dynamics distillation. In *NeurIPS*, 2022.