Model-based language-instructed reinforcement learning

Anonymous ACL submission

Abstract

We explore how we can build accurate world models which are partially specified by language and how we can plan with them in the face of novelty and uncertainty. We propose the first Model-Based Reinforcement Learning approach to tackle the environment Read To Fight Monsters (Zhong et al., 2019), a grounded policy learning problem. In RTFM an agent has to reason over a set of rules and a goal, both described in a language manual, and the observations, while taking into account the uncertainty arising from the stochasticity of the environment, in order to generalize successfully its policy to test episodes. We provide a sample-efficient proof-of-concept of the model-based approach for the basic dynamic task of RTFM. Furthermore, we show that the main open challenge of RTFM is learning the language-dependent reward function and suggest that future research should focus primarily on that task.

1 Introduction

Intelligent agents have the ability of re-composing known concepts to draw conclusions about new problems and this translates into the acquisition of very robust and general behaviours. Current Reinforcement Learning (RL) agents typically lack this ability and they need to be re-trained for every new problem; in contrast language models exhibit great generalization abilities, to the point that Large Language Models (LLMs) are increasingly considered foundation models (Bommasani et al., 2021), which can be pre-trained once on large corpora of text and re-used on any downstream language task with very little fine-tuning (Devlin et al., 2018; Brown et al., 2020; Chowdhery et al., 2022). Thus language-conditioned RL is a flourishing area of research.

On the other hand, language models are trained exclusively on textual inputs and struggle to ground the meaning of the words to real world dynamics. Multiple interactive environments have been proposed as a testbed for learning how to ground language (Chevalier-Boisvert et al., 2018; Zhong et al., 2019; Ruis et al., 2020; Küttler et al., 2020). While prior work mostly focuses on Behavioural Cloning or model-free RL, we argue for a Model-Based Reinforcement Learning (MBRL) approach, as this effectively decouples the problem of understanding how the world works from the problem of acting optimally in the world in order to solve one or more tasks. Concretely MBRL inherits the advantages of model-free RL of learning from scratch or from sub-optimal behaviour, while being orders of magnitude more sample efficient than the model-free counterpart. Furthermore it has the added value of being more interpretable and explainable. In fact, a decision made by a MBRL agent can be accompanied by human-interpretable examples of likely future trajectories that are taken into account by the model in making such a decision.

In this work, we focus on Read To Fight Monsters (RTFM), a challenging benchmark for testing grounded language understanding in the context of reinforcement learning proposed by Zhong et al. (2019). RTFM tests the acquisition of complex reading skills in RL agents in order to solve completely new tasks based on written descriptions of the task dynamics and goal. Critically, the written information provided is not enough on its own to obtain an optimal policy, but the agent needs to cross-reference multiple times such information with the current state of the environment in order to figure out a plan of action.

In this work, we make the following contributions: first, we formulate a language-instructed MBRL method for solving RTFM and show how to train an agent in this environment (see Fig. 1). Our method explicitly models the stochastic changes in the discrete environment and performs planning with a stochastic variant of Monte Carlo Tree Search (MCTS, Kocsis and Szepesvári, 2006). We

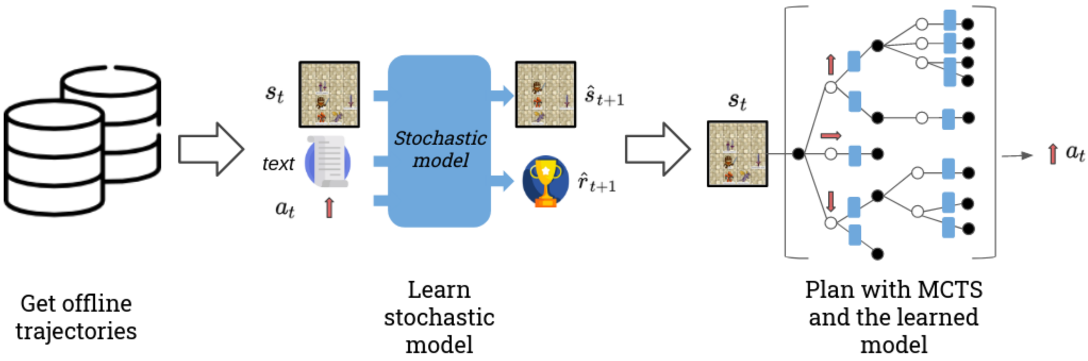

Figure 1: High-level view of the proposed method. We collect trajectories in the environment with behavioural policies, then use them to learn a discrete stochastic model of the environment and finally deploy the learned model at test time to plan with Monte Carlo Tree Search (MCTS).

then demonstrate performance compatible with the SOTA agent from Zhong et al. (2019), while using 150x less data[1] in the basic dynamic version of the RTFM environment. Furthermore, we highlight how predicting the reward accurately is critical for scaling the approach to more complex variants of the task, by showing a strong positive correlation between the reward accuracy and the win rate in different scenarios. Finally, we show that current neural architectures, based on CNNs and FiLM (Perez et al., 2018) or on transformers (Vaswani et al., 2017), are not able to learn the optimal reward function in the sample-efficient regime of 200k samples of terminal transitions for any task whose manual is written in rich natural language.

2 Related Work

Language Grounding and Understanding

Chevalier-Boisvert et al. (2018) proposes BabyAI, a benchmark for studying the sample efficiency of Imitation Learning and RL methods in tasks where the goal is specified in natural language. Ruis et al. (2020) instead studies the problem of compositional generalization in situated Language Understanding in a Supervised Learning setup with the gSCAN benchmark, where agents have to map language instructions to corresponding action sequences. Narasimhan et al. (2018) considers a transfer learning setup between pairs of grid-world environments, where entities are annotated with language information about their role and behaviour. Bahdanau et al. (2018) learns how to train reward models from language specifications and expert trajectories and shows the usefulness of such reward models in training RL agents to accomplish language specified tasks.

[1]We use only 1M frames while SOTA agent is trained with 150M frames in total.

Our work builds on the environment RTFM, introduced in Zhong et al. (2019), with the main target of solving such environment with a model-based approach instead of a model-free one. Similar work on grounding language can be found in Hanjie et al. (2021), which introduces the MESSENGER environment; a notable difference between RTFM and MESSENGER is that in the latter the co-reference of the entities and their names is harder to learn, but the reasoning steps to perform are easier. Zhong et al. (2021) proposes SILG, a unified interface for RTFM, MESSENGER, NetHack (Küttler et al., 2020) and symbolic abstractions of ALFRED (Shridhar et al., 2020) and Touchdown (Chen et al., 2019); each environment poses its own unique challenges, like learning multi-hop reasoning or grounding co-references, dealing with partial observability, large action spaces or rich natural language instructions and annotations. Both the baseline in Zhong et al. (2021) and the following work on SILG in Zhong et al. (2022) include in the benchmark only the simplest, stationary variation of RTFM and focus instead on finding model-free algorithms that are able to deal with all 5 SILG environments.

In this work, we focus only on RTFM and consider all the stochastic levels of the game, similarly to Zhong et al. (2019), and we propose the first model-based approach for this environment.

Model-based Reinforcement Learning

AlphaGo (Silver et al., 2016) is the first work demonstrating SOTA performance of MBRL with a MCTS-based agent which has access to the true simulator of the game of Go and learns with neural networks both a prior over promising actions and an evaluation function to estimate the values of game configurations. MuZero (Schrittwieser et al., 2019) lifts the constraint of having access to a simulator of the environment, by learning a latent model of it and using it to perform a variant of MCTS in the latent space with the aid of a value function and a policy.

In this work, for simplicity we do not use policy and value functions as it is not our focus, but they could be beneficial to reduce the simulation budget of our MCTS agent further and they would certainly be necessary to scale up this approach to higher dimensional action-spaces and longer-horizon tasks. Overall our contribution is orthogonal to the learning of policy and value networks for MCTS algorithms, as we aim to learn a good model of a stochastic environment and a complex language-dependent reward function that is able to generalize to new environments.

Most works in MBRL assume a deterministic environment (as it is the case for example in chess and Go) or weakly stochastic (as Atari) and show dramatic drops in performance when applied to stochastic ones. Ozair et al. (2021) demonstrates how MuZero performance deteriorates when playing chess if the opponent is considered part of the environment (version of chess denoted *single player*) and the algorithm cannot enumerate its actions, but has to learn to model them as possible stochastic outcomes.

The Vector Quantized Model (VQM) in Ozair et al. (2021) probably has the most similar approach to ours, learning a "State VQVAE" to extract discrete latent codes and then learning a "Transition model" which, given a latent state-action pair and a discrete latent code, produces the next latent state.

Another notable line of work capable of dealing with stochastic environments can be found in Hafner et al. (2018), Hafner et al. (2019) and Hafner et al. (2020) . These works are based on the Recurrent State Space Model (Hafner et al., 2018) and of particular interest is Hafner et al. (2020), as it also uses discrete latent variables to capture the stochasticity in the environment dynamics. The discrete variables are trained with straight-through gradients and the obtained model is used to produce synthetic data in the latent space to train a model-free algorithm instead of being used for planning. However, none of these models involves language.

Goal: `Fight the order of the forest.`
Manual: `Fire monsters are weak against gleaming items. Lightning monsters are defeated by grandmasters items. Use shimmering items for poison monsters. Rebel enclave has the following members: demon. Dragon are star alliance. Jinn are on the order of the forest team. Cold monsters are weak against blessed items.`
Inventory: `empty.`

Figure 2: Example of a frame from the RTFM environment with two monsters in the natural language version. Together with the grid observation (above), the agent is provided with the goal, manual and the inventory (below).

3 Read To Fight Monsters

Read To Fight Monsters (RTFM) is a challenging benchmark proposed by Zhong et al. (2019) for testing grounded language understanding in the context of RL. RTFM tests the acquisition of complex reading skills in RL agents in order to solve completely new tasks based on written descriptions of the task dynamics and goal.

Crucially, it is not enough to consult the written information in order to obtain an optimal policy, but the agent needs to perform a multi-step reasoning between such information and the current state of the environment in order to figure out a plan of action.

To elucidate the reasoning steps and reading skills needed to win an episode, we go through the concrete example reported in Fig. 2.

1. From the goal extract which team to defeat (`order of the forest`).

2. Search in the manual which monster is assigned to that team (`jinn`).

3. Find in the map the element type of the target monster (`fire`).

4. Search in the manual which modifier beats the target monster's type (`gleaming`).

5. Find in the map the item with the correct modifiers (`gleaming sword`).

6. Pick up the correct item (`gleaming sword`).

7. Engage the correct monster (`fire jinn`) in combat with the correct item (`gleaming sword`).

The agent is given a reward of $+1$ if it engages the correct monster in combat while carrying the correct item, -1 in any other encounter with a monster and a reward of 0 for all intermediate steps.

As every episode contains a procedurally generated set of (monster, element) pairs, (item, modifier) pairs, goal and manual entries, the agent cannot solve new episodes memorizing what is the right pair of item to take and monster to fight, but it has to learn to read the goal and the manual and cross-reference them with the environment observation. The agent's performance is tested on episodes generated in such a way that no assignments of monster-team-modifier-element are ever seen during training, to test whether the agent is able to generalize via reading to new environments with unseen dynamics.

We consider two variants of the original RTFM, the dynamical version with simple language `sl` and the natural language dynamical version `nl`; these correspond respectively to `dyna` and `dyna+nl` in the notation used by Zhong et al. (2019).

There are two differences between `sl` and `nl` tasks and they both concern the way in which the manual and the goal are expressed. The first difference is that `sl` uses fixed language templates like `"gleaming beats fire"` instead of one of multiple crowd-sourced natural language reformulations, like `"fire monsters are weak against gleaming items"`. The second difference is that in `sl` the sentences of the manual are always ordered in a specific way (e.g. the first sentence always refers to monsters of the `cold` element and the last sentence to which monster is part of the `star alliance`), whereas in the `nl` task the order of the sentences is always shuffled. We find the importance of this second point to be underappreciated in Zhong et al. (2019) and we introduce a new variant of the task named `nl + no shuffle` where we ablate the shuffling factor, in order to disentangle this factor from the natural language one.

For more in depth description of the environment and how it is generated the reader can refer to Zhong et al. (2019).

During training we modify the environment such that in the terminal transitions, when the agent interacts with a monster, the entity that has been defeated is not removed from the terminal state. However, the trained agent during the evaluation procedure doesn't need to use the modified environment, as terminal states are used only to train the representation encoder, whereas the transition model never takes them as input.

4 Language-conditioned world model

The goal of any RL agent is to find a policy $\pi(a|s)$ that maximizes the expected cumulative reward $\mathbf{E}_\pi[\sum_{t=0}^T R_t]$ received from the environment in an episode if all actions are taken according to such policy. In this work, we take the model-based approach to RL: we learn a language-conditioned stochastic model of the environment and use it to plan with a stochastic version of vanilla MCTS.

We name our method as Reader (for REinforcement learning Agent for Discrete Environments with wRitten instructions)[2]. It is composed of a world model and uses MCTS as its planning algorithm. The world model consists of three components (see Fig. 3).

1. the representation model which encodes the grid-world observations s_t into discrete codes z_t:
$$p(z_t \mid s_t, s_{t-1}, a_{t-1}),$$

2. the transition model that predicts the next state s_{t+1}:
$$p(s_{t+1} \mid s_t, a_t, m),$$

3. the reward model predicts the current-step reward:
$$p(r_t \mid s_{t-1}, s_t, a_{t-1}, m, g).$$

Note that in contrast to existing model-based agents, our world model is conditioned on the

[2]Inspired by Dreamer (Hafner et al., 2019).

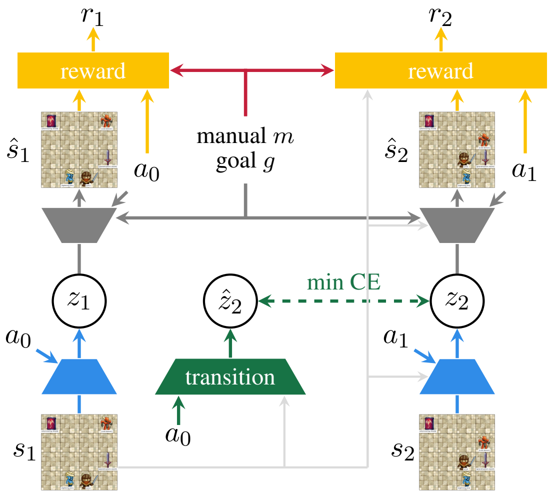

Figure 3: Components of language-conditioned world model.

episode-specific textual descriptions: the manual m which describes the roles of the monsters and the rules of the episode and the the sentence g describing the agent's goal.

4.1 Representation model

Our representation model is based on vector-quantized variational autoencoders (VQVAE, van den Oord et al., 2017), a latent variable model with discrete latent codes. The VQVAE architecture is composed of an encoder, a decoder and a vector quantization layer in between. Similarly to (Ozair et al., 2021), we condition the representation z_t of the current state s_t on the previous state s_{t-1}:

$$z_t = f(s_t, s_{t-1}, a_{t-1})$$

where z_t is a discrete code produced by the encoder f. The decoder d is trained to reconstruct the original state from the discrete code z_t given the previous state and action:

$$\hat{s}_t = d(z_t, s_{t-1}, a_{t-1}).$$

The discrete codes are produced by the encoder in the same way it was done in the original VQ-VAE paper (van den Oord et al., 2017) by keeping a trained codebook of prototypes vectors and selecting as a representation the prototype with the smallest distance to a continuous encoder output. We train the model end-to-end using the straight-through approximation for the vector quantization function when back-propagating through it. We use the three losses that were proposed by van den Oord et al. (2017) and implement the encoder and the decoder using a transformer architecture (Vaswani et al., 2017).

Since the RTFM environments has two sources of stochasticity (which correspond to two monsters), the quantization layer of our VQVAE produces two codes z_1, z_2, such that the continuous output of the encoder is split into parts and the quantization is performed for the two parts independently. This choice gives a combinatorial inductive bias to the representations we are learning.

4.2 Transition model

The purpose of the transition model $p(s_{t+1} \mid s_t, z_t, a_t, m)$ is to predict possible values of the next state s_{t+1} given the current state and the taken action. We implement the model by using an additional block (the green block in Fig. 3) which predicts the discrete representation z_{t+1} of the next state s_{t+1}

$$p(z_{t+1} \mid s_t, a_t)$$

We condition the transition model only on the manual m of the episode but not on the goal g, as the goal of the episode affects only the reward function. To simulate the next state s_{t+1} at the planning stage, the output of this transition block is passed though the decoder of the representation model.

We train the transition model concurrently with training the representation model, using the codes produced by the VQVAE as the model targets. We stop the gradients such that the existence of the transition model does not affect the learned representations. We use a transformer architecture for the transition model.

4.3 Reward model

A key aspect of RTFM is to model correctly the language-instructed reward function. Since the environment is stochastic, a natural choice for the reward function is $p(r_{t+1} \mid s_t, s_{t+1}, a_t, m, g)$, as including the next state s_{t+1} lets us predict a different reward for every possible stochastic outcome. This is only possible because we can predict the next state s_{t+1} with our stochastic transition model.

We train the reward model concurrently with the other two models and during training we use the true next state for s_{t+1}. The main neural architecture that we consider is a transformer. For additional studies, we also consider as an ablation an architecture comprising CNN and FiLM layers, which draws inspiration from the txt2π actor-critic architecture proposed in Zhong et al. (2019), but processes a_t and s_{t+1} as additional inputs and predicts the distribution of the reward r_{t+1}, instead of the policy and value of the actor-critic case. We refer to the models using this architecture as CNN+FiLM.

With the same model we also predict if the next state is terminal or not and what are the legal actions that can be taken in the next state. Both of these predictions are trivial, as they do not depend on the language.

Furthermore, since RTFM gives non-zero rewards only for terminal transitions, empirically we find beneficial in terms of sample efficiency and performance to train the reward function only on those transitions. For planning, we first predict if a transition is terminal or not; if the transition is non-terminal, we assign a reward of 0, if it is terminal, we predict the reward with the reward model.

4.4 Transformer baseline world model

A possible alternative is to have a model learn directly the joint distribution over s_{t+1} and r_{t+1} auto-regressively:

$$p(s_{t+1}, r_{t+1}|c_t) = p(r_{t+1}|s_{t+1}, c_t) \cdot \prod_{ij=1}^{H*W} p(s_{t+1,ij}|s_{t+1,<ij}, c_t), \quad (1)$$

where $c_t = (s_t, a_t, \texttt{wiki}, \texttt{goal})$ is the conditional information available, using for example a single Transformer model (Vaswani et al., 2017). While this model is possibly more general and it can be trained in teacher-forcing mode to capture correctly the stochasticity of the environment, its auto-regressive modeling over the entire next state makes it impractical for planning, where a model can be used hundreds if not thousands of times during planning for every single time-step. We nonetheless consider this model as a baseline, dubbed "Transformer baseline".

5 MCTS planning with the learned world model

To use the model for planning, we need to predict the next state s_{t+1} and reward r_{t+1} given the information available at the current step, that is s_t, a_t, m and g. We do that by first using the transition model to sample the next discrete latent code $\hat{z}_{t+1}$, then using the decoder to get $\hat{s}_{t+1}$ and finally the reward model to get $\hat{r}_{t+1}$.

To extend MCTS to stochastic transitions, we use two types of nodes in the tree: state nodes and state-action nodes. The tree branches at state nodes by considering different actions and it branches at state-action nodes by looking at different stochastic outcomes. While we have control over the actions that we choose, the outcomes and corresponding rewards are always sampled by using the learned model. We index the possible outcomes from a state-action node based on the indices of the pair of discrete codes (z_1, z_2) sampled by the transition model; if the codes have been already sampled, we continue traversing the already-expanded tree, otherwise we expand the newly-sampled state node.

6 Experiments

In this section, we conduct experiments to answer the following research questions: What is the accuracy and data efficiency of the MBRL method in the RTFM environment, and how does that compare with existing model-free approaches? What is currently the limiting factor that needs to be addressed when scaling MBRL sample-efficiently to more complex RTFM tasks?

To address these questions, we consider the RTFM environment of two levels of complexity: first, a basic version that uses simplified language (`sl`) is used to validate the model and compare it with alternatives; second, a more complex version that uses full natural language (`nl`) is used to investigate the limits and scalability of the current MBRL solution. We do not present results for many-to-one entities assignment tasks (referred in Zhong et al. (2019) as `dyna+groups` and `dyna+groups+nl`), as empirically we find that the gap in performance from `sl` to `dyna+groups` much smaller than from `sl` to `nl` and we hypothesize that simply scaling up the resources allocated to the task would be enough to solve it optimally. We furthermore consider the natural language variant without the wiki shuffling `nl + no shuffle` described in Sec 3.

6.1 Training

To ease the computational demands of the full RL pipeline, we consider an offline RL setup. We first collect a dataset of trajectories for each task, then train the model on it without ever directly interacting with the real environment and then evaluate the MCTS agent equipped with the learned model by playing 1000 episodes in the test environment.

To collect the datasets we use a random policy for 50% of the episodes and a vanilla MCTS policy[3] with access to a simulator of the real environment for the remaining 50% of the trajectories. Each dataset is composed of 200k episodes, or roughly 1M frames; full details are reported in Table 3 in the appendix.

We train the model with mini-batches of 1-timestep transitions sampled from the dataset and train the model components as described in Sec 4. Following Zhong et al. (2019), we optionally make use of a curriculum to train the model on the harder variations of the environment. Specifically we first train from scratch over `sl`, then we continue training the model on `nl` and `nl + no shuffle`; when doing so, we add `+ curriculum` to the name of the task.

6.2 Results

In the first experiment we train our Reader model on the `sl` task and evaluate its performance with MCTS, using 400 simulations per time step. We compare against two strong agents: the first, which we call **Oracle MCTS**, uses the same MCTS algorithm of our agents, but instead of using a learned model of the environment, it is using the ground-truth simulator; this represents an upper bound for the performance of our agent, since the planning algorithm is identical and the model is perfect; the second is the model-free agent txt2π proposed in (Zhong et al., 2019), which detains the state-of-the-art in RTFM environment. We also compare to our model-based Transformer baseline described in Sec. 4.4; this model is also evaluated with MCTS, but we use only 100 simulations per time step, as planning with a full auto-regressive transformer model is much slower than planning with Reader.

We report the results in Table 1. Reader achieves results comparable with model-free SOTA agent txt2π from (Zhong et al., 2019), while using only 1M frames instead of the 150M used to train txt2π. Furthermore, Reader is better than the Transformer baseline, both in terms of win rate and wall-clock time for planning and comes close to the performance of the Oracle MCTS agent. This result validates our method and serves as a proof-of-concept for sample-efficient MBRL in language-instructed environments.

[3]We restrict the computational budget per action of the MCTS policy in such a way that it is not optimal.

Models	Win rate
txt2π (Zhong et al., 2019)	0.85 (0.09)
Reader	0.82 (0.02)
Transformer baseline	0.76 (0.05)
Oracle MCTS	0.85 (0.01)

Table 1: **Comparison of methods (`sl`)**. We evaluated every model with 5 independent training runs and report the average and standard deviation (over the 5 runs) of the win rate over 1000 episodes. The Reader and the Transformer baseline were trained with 1M frames. The txt2π was trained for 100M frames on a static version of the `sl` task and then for another 50M frames on `sl`; the reported results are from (Zhong et al., 2019).

In our second experiment we show how the test accuracy of the reward function is predictive of the win rate of the agent.

We define 5 mutually exclusive logical cases that fully partition the set of terminal transitions; the first case is when the agent interacts with any monster while having an empty inventory and the other 4 cases are all the combinations of interacting with the right or wrong monster while carrying the right or the wrong item in the inventory. The reason behind this choice for the metric is that it re-weights the accuracies over different transitions, making the metric more robust against distribution shifts between the offline test set (collected by behavioural policies) and the set of trajectories produced by the trained agent during its online evaluation.

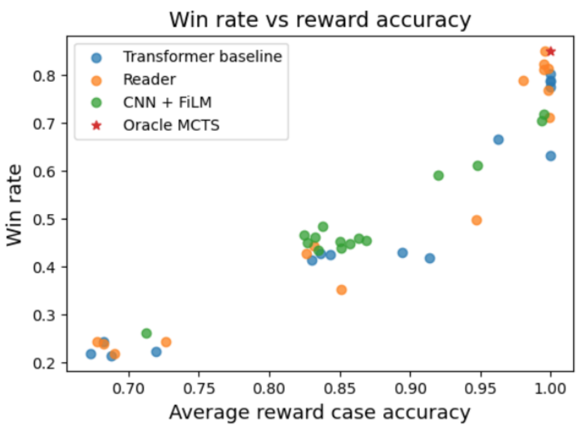

Figure 4: **Importance of reward accuracy.** We train 5 random seeds for the Reader model, the Transformer baseline and the CNN + FiLM ablation for the `sl`, `nl + curriculum` and `nl + shuffle + curriculum` tasks. We plot the win rate as a function of the average reward case accuracy for each individual run.

For each case of terminal transitions, we com-

Models	sl	nl	nl + curriculum	nl + no shuffle + curriculum
Transformer	0.99 (0.01)	0.71 (0.01)	0.79 (0.06)	0.96 (0.07)
CNN + FiLM	0.85 (0.02)	0.69 (0.03)	0.84 (0.01)	0.86 (0.05)

Table 2: **Average reward case accuracy**. We split the terminal transitions in the test set in 5 mutually exclusive logical cases in which the terminal transitions can be classified; we compute the accuracy over each case and then take the average. We evaluate every model with 5 independent training runs and report the average and standard deviation (over the 5 runs) of the accuracy.

pute the reward accuracy, then we compute the average among the cases and call this measure "average reward case accuracy". We train the Reader agent, the Transformer baseline agent and an ablation of Reader using the CNN+FiLM reward model on the environments `sl`, `nl + curriculum` and `nl + no shuffle + curriculum` and report the win rate of individual runs as a function of the average reward case accuracy in Fig. 4. Our experiment shows how, regardless of the model (Reader or Transformer baseline) and regardless of the reward model architecture (transformer or CNN+FiLM), the average reward case accuracy is strongly and positively correlated with the win rate of the agent.

Finally we study the impact of model on reward prediction accuracy by training the transformer reward model and the CNN+FiLM reward model on datasets of 200k samples containing only the terminal transitions for the environments `sl`, `nl`, `nl + curriculum` and `nl + no shuffle + curriculum` and report the results in Table 2.

We find that both the transformer and the CNN+FiLM architectures are not able to generalize to the test set for the tasks featuring natural language. While training via curriculum from the `sl` environment helps, we notice that in `nl` a good part of the difficulty is due to the shuffling of the sentences in the manual. The transformer architecture trained on `nl + no shuffle + curriculum` mostly retains the accuracy achieved in `sl`, while this is not the case for the CNN+FiLM model. This experiment highlights how the reward prediction task is the current bottleneck in solving the RTFM tasks with natural language and suggests that focusing on a Supervised Learning setup for the reward prediction and using a dataset of terminal transitions are promising approaches to isolate the core challenge of the RTFM environment.

7 Discussion and Conclusions

RL environments such as RTFM are great for developing models capable of natural language understanding. Prior work shows that the model-free approach might work well in those benchmarks, but it requires a large number of samples in order to work and it is not insightful with regard to the agent's understanding capabilities. On the other hand the model-based approach is a promising one for multiple reasons: first one can leave aside the exploration problem, which might not be crucial for simple environments such as RTFM, by using offline experience collected from simple behavioural policies. Second, one can decouple the planning task from the task of model building and even that can be split in multiple components. This lets us for example learn independently the stochastic dynamics and the reward model, which we show is a crucial component for a strong agent in RTFM.

For RTFM, we observe that a model's accuracy in the reward prediction task correlates well with the final performance in the RL task. Thus, in order to understand the limitations of the existing architectures applicable to grounded language learning it makes sense to focus on the reward prediction task. Our experiments show that existing architectures (built on CNN+FiLM and transformers) need a large number of samples to learn the RTFM reward. For example, the transformer model has the tendency to overfit to positional information: the performance drops significantly with the permutation of sentences. Therefore, more research is needed to improve the sample efficiency of models for grounded language learning; leveraging the out-of-the-box representation learning capabilities of pre-trained large language models seems a promising direction to explore.

References

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. 2018. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258.

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

Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. 2019. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12538–12547.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2018. Babyai: A platform to study the sample efficiency of grounded language learning.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2019. Dream to control: Learning behaviors by latent imagination.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2018. Learning latent dynamics for planning from pixels.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2020. Mastering atari with discrete world models.

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

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. volume 4212 LNAI, pages 282–293. Springer Verlag.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. 2020. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684.

Adrian Lancucki, Jan Chorowski, Guillaume Sanchez, Ricard Marxer, Nanxin Chen, Hans J. G. A. Dolfing, Sameer Khurana, Tanel Alumäe, and Antoine Laurent. 2020. Robust training of vector quantized bottleneck models. *CoRR*, abs/2005.08520.

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

Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron Van Den Oord, and Oriol Vinyals. 2021. Vector quantized models for planning.

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

Proceedings of the AAAI Conference on Artificial Intelligence, volume 32.

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

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. 2019. Mastering atari, go, chess and shogi by planning with a learned model.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. *CoRR*, abs/1711.00937.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Victor Zhong, Austin W Hanjie, Sida Wang, Karthik Narasimhan, and Luke Zettlemoyer. 2021. Silg: The multi-domain symbolic interactive language grounding benchmark. *Advances in Neural Information Processing Systems*, 34:21505–21519.

Victor Zhong, Jesse Mu, Luke Zettlemoyer, Edward Grefenstette, and Tim Rocktäschel. 2022. Improving policy learning via language dynamics distillation. *arXiv preprint arXiv:2210.00066*.

Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. 2019. RTFM: generalising to novel environment dynamics via reading. *CoRR*, abs/1910.08210.

A Offline trajectories datasets

To collect the datasets we use a random policy for 50% of the episodes and a vanilla MCTS policy with access to a simulator of the real environment for the remaining 50% of the trajectories.

The MCTS policy used to collect data specifically uses 30 internal simulations per time-step, maximum rollout length of 10, a discount factor of 0.9 and the Upper Confidence Bound constant $c = 1$.

We report in Table 3 the dataset statistics for the `sl` environment. Since all other dynamical variants of RTFM have the same underlying mechanics (e.g. 2 monsters and 2 items placed randomly in a grid world, the monsters move stochastically according to an unknown policy which is always the same in all the variants), all the dataset statistics are identical, up to stochastic fluctuations, for `nl` and `nl + no shuffle`.

Dataset	**sl**
Tot. frames	971k
Non-terminal	771k
Successes	77k (8%)
Failures	123k (13%)
Win rate (random)	4.6%
Win rate (MCTS)	71.0%

Table 3: Datasets collected for offline training of the model and relative statistics.

B Training details

We report in the following section the summary details about the architectures and training procedures used in this work.

Encoder and Decoder of the representation model: two transformer encoder layers each, model dimension of 128, feed-forward dimension of 256, gelu activation function, dropout of 0.1.

Transition model: two transformer encoder layers and two transformer decoder layers, model dimension of 128, feed-forward dimension of 256, gelu activation function, dropout of 0.1.

Reward model (transformer): six transformer encoder layers, model dimension of 256, feed-forward dimension of 1024, gelu activation function, dropout of 0.1.

Reward model (CNN+FiLM): embedding dimension of 30, dimension of small RNN of 10, dimension of Bi-LSTM of 100 and dimension of final representations of 400. For more details about the architecture, please refer to (Zhong et al., 2019).

Vector Quantization layer: 2 codebooks of 32 codes with feature dimension 64 (half of the model dimension of the encoder and decoder model dimension). We use a commitment loss coefficient

$\beta = 0.25$ and a codebook learning rate multiplier $\lambda = 5$. Following (Lancucki et al., 2020), we use KMeans ++ to reinitialize the codes during the first 50 epochs, once every 50 forward passes.

Transformer baseline: we use a encoder-decoder architecture, plus the reward model from Reader. Encoder and decoder use two layers, the reward model six. All layers use model dimension of 256, feed-forward dimension of 1024, gelu activation function, dropout of 0.1.

We report the other hyper-parameters used during training in Tab 4.

Hyper-parameters	**Values**
Batch size	500
Optimizer	Adam
Learning rate	10^{-4}
Lr warm-up steps	400
Epochs	250

Table 4: Hyper-parameters used.

C Computational resources used

GPU resources used for Tab. 1 and Fig 4 are reported in Tab 5, whereas the CPU resources for the MCTS evaluation of the agents are reported in Tab 6.

Model	**single (h)**	**total (h)**
Reader	23	345
Transformer baseline	20	300
CNN+FiLM	4.3	65
Total		710

Table 5: GPU resources used for Tab. 1 and Fig 4. We use 5 random seeds and train on 3 tasks.

Model	**single (h)**	**total (h)**
Reader (20)	2.5	750
Transformer baseline (40)	5	3000
CNN+FiLM (20)	1.5	450
Total		4200

Table 6: CPU resources used for Tab. 1 and Fig 4. We use 5 random seeds and evaluate on 3 tasks. Number of CPU cores used for every run is reported between parenthesis after the name of the model.

The resources used for Tab. 2 are reported in Tab. 7.

All experiments use the Tesla V100 GPU model and the total amount of resources used to obtain the reported results is approximately 1000 GPU hours and 4200 CPU hours.

Model	**single (h)**	**total (h)**
Transformer	10	200
CNN+FiLM	4.3	86.6
Total		286.6

Table 7: GPU resources used for Tab. 2 . We use 5 random seeds and train on 4 tasks.