Learning to Follow Instructions in Text-Based Games

Anonymous Author(s)
Affiliation
Address
email

Abstract

Text-based games present a unique class of sequential decision making problem in which agents interact with a partially observable, simulated environment via actions and observations conveyed through natural language. Such observations typically include instructions that, in a reinforcement learning (RL) setting, can directly or indirectly guide a player towards completing reward-worthy tasks. In this work, we study the ability of RL agents to follow such instructions. We conduct experiments that show that the performance of state-of-the-art text-based game agents is largely unaffected by the presence or absence of such instructions, and that these agents are typically unable to execute tasks to completion. To further study and address the task of instruction following, we equip RL agents with an internal structured representation of natural language instructions in the form of Linear Temporal Logic (LTL), a formal language that is increasingly used for temporally extended reward specification in RL. Our framework both supports and highlights the benefit of understanding the temporal semantics of instructions and in measuring progress towards achievement of such a temporally extended behaviour. Experiments demonstrate the superior performance of our approach.

1 Introduction

Building AI agents that can understand natural language is an important and longstanding problem in AI. In recent years, instrumented text-based game (TBG) engines have served as compelling environments for studying a variety of tasks related to language understanding, affordance extraction, memory, and sequential decision making (e.g., Côté et al. (2018); Adhikari et al. (2020); Liu et al. (2022)). They provide a simulated, partially observable environment where an agent can navigate and interact with environment objects, receiving observations and administering commands via natural language. TextWorld (Côté et al., 2018) is a TBG learning environment for training reinforcement learning (RL) agents. Successful play requires language understanding, effective navigation, memory, and an ability to follow instructions embedded within the text. Instructions may or may not be directly bound to reward but can guide an RL agent towards completing tasks and collecting reward.

In this paper we study instruction following in text-based games and propose an approach that advances the previous state of the art. To this end, we employ the state-of-the-art model-free TBG RL agent called GATA (Graph Aided Transformer Agent) (Adhikari et al., 2020) that operates in the TextWorld environment. GATA has made significant advances in performance by augmenting TBG agents with long-term memory – a critical component of effective game play. Despite GATA's improvement over previous baselines, our experiments (see Figure 1) show that GATA performance is largely unaffected by the presence or absence of instructions, leading us to conclude that GATA is not effectively following instructions. We also find that while GATA agents are able to garner reward,

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

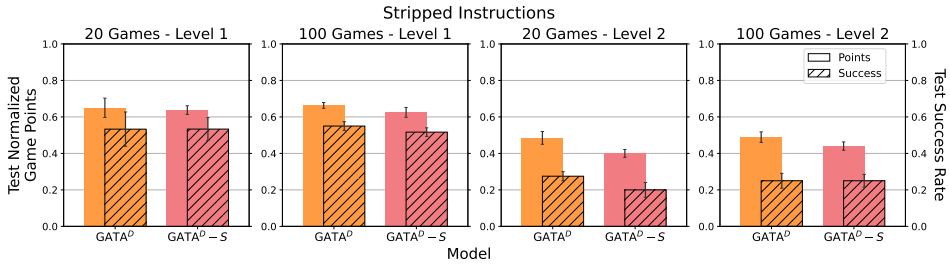


Figure 1: Comparison of GATA performance when trained with instructions (GATA^{D}) versus when instructions are stripped from environment observations (GATA^{D}-S). Agents were trained with 20 or 100 games, at increasing levels of task difficulty (level 1 vs level 2). Note that normalized game point performance (solid blocks) and rate of success (hashed blocks) are largely unchanged whether instructions are present or absent. Low success rate (i.e., task completion) rate is also seen in level 2.

they are not typically successful in *completing* tasks – an important vulnerability to the deployment of such techniques in environments where partial completion of tasks can be unsafe.

To further study and address the task of instruction following, we equip GATA with an internal structured representation of natural language instructions specified in Linear Temporal Logic (LTL) (Pnueli, 1977), a formal language that is increasingly used for temporally extended goals in planning and reward specification in RL. LTL also provides a mechanism to monitor progress towards completion of instructions. Our framework both supports and highlights the benefit of understanding the temporal semantics of instructions and in measuring progress towards achievement of a temporally extended behaviour. We perform experiments that illustrate the superior performance of our TBG agent and its ability to follow instructions. Contributions of this work include:

- Experiments that expose the lack of instruction following and low task completion rate in a state-of-the-art TBG agent.
- An approach to the study and deployment of instruction following in TBG environments via exploitation of a formal language: LTL. LTL provides well-defined semantics and supports a measure of progress towards satisfaction of instructions.
- An augmentation to an existing state-of-the-art architecture for TBGs to equip a TBG agent with instruction-following capabilities.
- Comprehensive experiments and insights that study our and others' approaches to instruction following, and that highlight the superior performance of our proposed approach.

2 Background

In this section we introduce TextWorld, the TBG engine that we use, together with the Cooking domain that we employ in our experiments. We also overview Linear Temporal Logic, which (as described in section 1) we use in our approach as an internal representation for instructions.

2.1 Text-Based Games: TextWorld

Text-based games are partially observable multi-turn games where the environment and the player's action choices are represented textually. In this work, we use TextWorld (Côté et al., 2018) as our text-based game engine. A text-based game can be viewed as a (discrete-time) partially observable Markov decision process (POMDP) $\langle S, T, A, O, \Omega, R, \gamma \rangle$ (Côté et al., 2018) where S is the environment's state space, A is the action space, $T(s_{t+1}|s_t, a_t)$ where $s_{t+1}, s_t \in S$ and $a_t \in A$ is the conditional transition probability between states s_{t+1} and s_t given action a_t, O is the set of (partial) observations that the agent receives, $\Omega(o_t|s_t, a_{t-1})$ is the set of conditional observation probabilities, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. An agent's goal is to learn some optimal policy $\pi^*(a|o)$ (or a policy that conditions on historical observations or on some internal

memory) that maximizes the expected discounted return. In this work, we focus on the choice-based variant of games, similar to previous works (Adhikari et al., 2020; Narasimhan et al., 2015). The action space A is a list of possible commands and at each time-step t in the game, the agent must select action $a_t \in C_t$ from the current subset of permissible actions $C_t \subset A$.

2.1.1 Environment Setting

We focus on the TextWorld *Cooking domain*, popularized by Adhikari et al. (2020) and Microsoft's First TextWorld Problems: A Language and Reinforcement Learning Challenge (FTWP) (Trischler et al., 2019). The game tasks agents with gathering and preparing various cooking ingredients described by an in-game recipe that is to be found. Game points (rewards) are earned for each of (1) collecting a required ingredient, (2) performing a preparatory step (some cutting or cooking action) on an ingredient as required by the recipe, (3) preparing the meal once all of the ingredients have been prepared, and (4) eating the meal. The game's partial observations can contain instructions that guide the agent towards completion of tasks, but not all instructions correspond directly to rewards. The game first instructs the agent to examine a cookbook, which elicits a recipe to be followed. The act of examining the cookbook returns no reward, but following its recipe will return reward. See Appendix C for more details. Success is determined by whether the recipe is fully completed and eaten. Preparing ingredients can also involve collecting certain tools (e.g., a knife). The game may also involve navigation – the agent may need to navigate to the kitchen or to find certain ingredients.

2.2 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) (Pnueli, 1977) is a formal language – a propositional logical language with temporal modalities – that can be used to describe properties of trajectories. We will use LTL to specify instructions. LTL formulas are constructed from propositional variables (e.g., `player-has-carrot`), connectives from propositional logic (e.g. $\neg$), and two temporal operators: $\bigcirc$ (`NEXT`) and U (`UNTIL`). Formally, we define the syntax of LTL per Baier & Katoen (2008) as

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc \varphi \mid \varphi \,\mathsf{U}\, \psi$$

where $p \in \mathcal{P}$ for some finite set of propositional symbols $\mathcal{P}$. Satisfaction of an LTL formula is determined by a sequence of truth assignments $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \ldots \rangle$ for $\mathcal{P}$, where $p \in \sigma_i$ iff proposition $p \in \mathcal{P}$ holds at time step i. Formally, σ *satisfies* φ at time $i \geq 0$, denoted as $\langle \sigma, i \rangle \models \varphi$, under the following conditions:

- $\langle \sigma, i \rangle \models p$ iff $p \in \sigma_i$, where $p \in \mathcal{P}$
- $\langle \sigma, i \rangle \models \neg\varphi$ iff $\langle \sigma, i \rangle \not\models \varphi$
- $\langle \sigma, i \rangle \models \bigcirc\varphi$ iff $\langle \sigma, i+1 \rangle \models \varphi$
- $\langle \sigma, i \rangle \models (\varphi \wedge \psi)$ iff $\langle \sigma, i \rangle \models \varphi$ and $\langle \sigma, i \rangle \models \psi$
- $\langle \sigma, i \rangle \models \varphi \,\mathsf{U}\, \psi$ iff there exists j such that $i \leq j$ and $\langle \sigma, j \rangle \models \psi$, and $\langle \sigma, k \rangle \models \varphi$ for all $k \in [i, j)$

A sequence σ is then said to *satisfy* φ iff $\langle \sigma, 0 \rangle \models \varphi$.

Any LTL formula can be defined in terms of $p \in \mathcal{P}$, $\neg$ (*negation*), $\wedge$ (*and*), $\bigcirc$ (`NEXT`), and U (`UNTIL`). From these operators, we can also define the Boolean operators $\vee$ (*or*) and $\rightarrow$ (*implication*), and the temporal operators $\Box$ (`ALWAYS`) and $\Diamond$ (`EVENTUALLY`), where $\langle \sigma, 0 \rangle \models \Box\varphi$ if φ always holds in σ, and $\langle \sigma, 0 \rangle \models \Diamond\varphi$ if φ holds at some point in σ.

2.2.1 LTL Progression

LTL formulas can also be *progressed* along a sequence of truth assignments (Bacchus & Kabanza, 2000; Toro Icarte et al., 2018b) In other words, as an agent acts in the environment, resulting truth assignments can be used to update the formula to reflect what has been satisfied. The updated formula would now reflect the parts of the original formula that are remaining to be satisfied or whether the formula has been violated/satisfied. The progression operator $\mathrm{prog}(\sigma_i, \varphi)$ is defined as:

Definition 2.1. For LTL formula φ, truth assignment σ_i over $\mathcal{P}$, and $p \in \mathcal{P}$, $\mathrm{prog}(\sigma_i, \varphi)$ is defined as

- $\mathrm{prog}(\sigma_i, p) = \begin{cases} \mathsf{true} & \text{if } p \in \sigma_i \\ \mathsf{false} & \text{otherwise} \end{cases}$
- $\mathrm{prog}(\sigma_i, \neg\varphi) = \neg\,\mathrm{prog}(\sigma_i, \varphi)$
- $\mathrm{prog}(\sigma_i, \mathtt{NEXT}\,\varphi) = \varphi$
- $\mathrm{prog}(\sigma_i, \varphi_1 \wedge \varphi_2) = \mathrm{prog}(\sigma_1, \varphi_1) \wedge \mathrm{prog}(\sigma_1, \varphi_2)$
- $\mathrm{prog}(\sigma_i, \varphi_1\,\mathtt{UNTIL}\,\varphi_2) = \mathrm{prog}(\sigma_1, \varphi_2) \vee (\mathrm{prog}(\sigma_1, \varphi_1) \wedge \varphi_1\,\mathtt{UNTIL}\,\varphi_2)$

In the context of TextWorld, the progression operator can be applied at every step in the episode to update the LTL instruction fed to the agent. To do so, it's necessary to have *event detectors* that can detect when propositions are true as the agent acts during an episode (e.g., to detect that `player-has-carrot` is true when the player has the carrot). We discuss how event detection occurs in section 4, and give an example of how progression works in Appendix D.

3 Following Instructions with GATA

In order to evaluate the effectiveness of state-of-the-art text-based game agents at following instructions, we conducted experiments on the Cooking domain using the state-of-the-art model-free RL agent for TextWorld, GATA (Adhikari et al., 2020). GATA uses a transformer variant of the popular LSTM-DQN (Narasimhan et al., 2015) combined with a dynamic belief graph that is updated during game-play. The aim is to use this belief graph as long-term memory to improve action selection by modelling the underlying game dynamics (Adhikari et al., 2020). Formally, given the POMDP, GATA attempts to learn some optimal policy $\pi^*(a|o, g)$ where g is the belief graph.

While GATA's belief graph can capture goal relations (e.g. apple-needs-cut), it turns out that agents trained to condition on observations and the GATA belief graph alone largely ignore in-game instructions. We tested a GATA agent on levels 1 and 2 in the Cooking domain, after training on either the 20-game or 100-game training set, and found that in none of those settings was the cookbook examined more than 15% of the time (3/20 testing games). In short, *the GATA agent usually doesn't observe what the recipe is for the current game*, meaning it has no way of knowing what the actual goal of the game is (except – eventually – from the rewards it gets and when the episode ends).

We further investigate how GATA agents fail to follow instructions by training these agents using modified game observations that have their instructions stripped (specifically, instructions directing the agent to examine the cookbook, the recipe text within the cookbook, and instructions to grab a knife if attempting to cut an ingredient without first holding the knife were removed from observations). This has two effects: (1) the agent no longer receives text-based instructions about what the goal is or what it should do; and (2) GATA's belief state will no longer capture goal relations like 'needs'. The results of this experiment are in Figure 1, and demonstrate how GATA's performance remains largely unchanged. This suggests that GATA is (here at least) (a) *not exploiting text-based instructions that would lead it to success* and (b) *even not exploiting the goal-related relations in its own belief state*.

The results in Figure 1 also show a drop in GATA's performance when moving from level 1 to level 2 in the Cooking domain, where the games' complexity is increased by just one added ingredient preparation step in the recipes (see Table 1 for more details on the levels). GATA has difficulty in fully completing tasks on level 2 games, where its success rate is roughly half that of its achieved normalized game points (only the latter metric was used by Adhikari et al. (2020)).

Given these insights, we wish to further study and address instruction following in TBGs. In the next section, we propose using LTL and demonstrate how existing work can be easily augmented.

4 An Approach to Following Instructions

We now investigate a mechanism for both studying and advancing the ability of an RL agent to follow instructions. We do so by translating instructions to an internal structured representation of language in the form of LTL, a formal language that is increasingly being used for reward specification in RL agents (Vaezipoor et al., 2021; Leon et al., 2020; Kuo et al., 2020; Camacho et al., 2019; Toro Icarte et al., 2018b). We describe how to augment the GATA architecture with these LTL instructions and how to monitor progress towards their completion.

4.1 Generating and Representing LTL Instructions for TextWorld

We use three types of instructions for the Cooking domain. The first instruction identifies the need to examine the cookbook: This instruction is defined as φ : `NEXT` `cookbook-is-examined`. This in-

struction simply states that the agent should examine the cookbook (i.e. `cookbook-is-examined` = true) in the next step of the game. The second instruction is the actual recipe that gets elicited from the cookbook. We format this instruction to be *order-invariant* and *incomplete*. Order-invariance allows the agent to complete the instructions in any order, but is still constrained by any ordering that the TextWorld engine may enforce. "Incomplete" simply refers to the fact that not every single action required to complete the recipe is encoded (i.e. grabbing a knife before slicing a carrot, opening the fridge). The agent must still learn to do these things to accomplish its tasks, but is not directly instructed to. Assuming the recipe requires that predicates $p_1, p_2, \ldots p_n$ be true, the cookbook instructions are modelled as $\varphi : (\texttt{EVENTUALLY}\, p_1) \wedge (\texttt{EVENTUALLY}\, p_2) \wedge \ldots (\texttt{EVENTUALLY}\, p_n)$.

For example, in the Cooking Domain, this instruction might be the conjunction

$$\varphi : (\texttt{EVENTUALLY}\, \texttt{apple-in-player}) \wedge (\texttt{EVENTUALLY}\, \texttt{meal-in-player}) \wedge (\texttt{EVENTUALLY}\, \texttt{meal-is-consumed}).$$

The final type of instruction identifies the need to navigate to the kitchen. This instruction is defined as $\varphi : \texttt{EVENTUALLY}\, \texttt{player-at-kitchen}$. This instruction will come prior to the first two described above, but is only used in games with navigation (see Table 1).

We build a simple LTL translator that generates these instructions from the textual observations, similar to the goal generator used in Liu et al. (2022). TextWorld's observations are easily parsed to extract the goal information already contained within them, which we then formalize and keep track of using LTL. We provide examples of these observations and more details in Appendix E. Note that these observations are only used to generate the instruction itself, and subsequently LTL progression is used with the GATA belief state as our event detector to monitor completion of instruction steps and to update instructions that remain to be addressed.

4.2 LTL Augmented Rewards and Episode Termination

We can also reward our agent for completing instructions, which we model as reward $R_\Phi(s, a, \varphi)$. For some labelling function $L : S \times A \to 2^{\mathcal{P}}$ that assigns truth values to the propositions in $\mathcal{P}$,

$$R_\Phi(s, a, \varphi) = R(s, a) + \begin{cases} 1 \text{ if } \operatorname{prog}(L(s, a), \varphi) = \mathsf{true} \\ -1 \text{ if } \operatorname{prog}(L(s, a), \varphi) = \mathsf{false} \\ 0 \text{ otherwise} \end{cases}$$

In other words, a bonus reward is given for every LTL instruction the agent satisfies and a penalty is given if the agent fails to complete an instruction. We perform an ablative study on the effect of this reward in subsubsection H.4.2. We henceforth refer to this modified reward function as the *LTL reward*. The maximum bonus reward an agent receives is either 2 if there is no navigation task, or 3.

Further, because we wish to *satisfy* instructions, we can also use the instructions to modify episode termination. That is, if our LTL instruction is violated, we have arrived in a terminal state, even if TextWorld has not indicated so. We perform an ablative study on the effect of this *LTL-based termination* in subsubsection H.4.2.

4.3 LTL-GATA Model Architecture

We build a similar model to GATA's original architecture, augmented to include the LTL encoding of instructions and their progression according to observed system state. We dub this model LTL-GATA, which we describe in detail below. Figure 2 depicts an episode step interaction of LTL-GATA with TextWorld and Table 3 depicts the model itself. Additional details can be found in Appendix F.

Graph Updater: We use the original GATA-GTP model (Adhikari et al., 2020), which generates a discrete belief graph as a list of triplets of the form $(\textit{object}, \textit{relationship}, \textit{object})$. It is composed of two sub-components: (a) the belief state updater, which generates g_t from observation o_t and the graph g_{t-1}; and (b) the graph encoder, which encodes the current graph into a vector as $\texttt{GE}(g_t) = g'_t \in \mathbb{R}^D$ for some latent dimension D. The graph encoder is a relational graph convolutional network (R-GCN) (Schlichtkrull et al., 2018) using basis regularization (Schlichtkrull et al., 2018) and highway connections (Srivastava et al., 2015). We refer the readers to Adhikari et al. (2020) for more details.

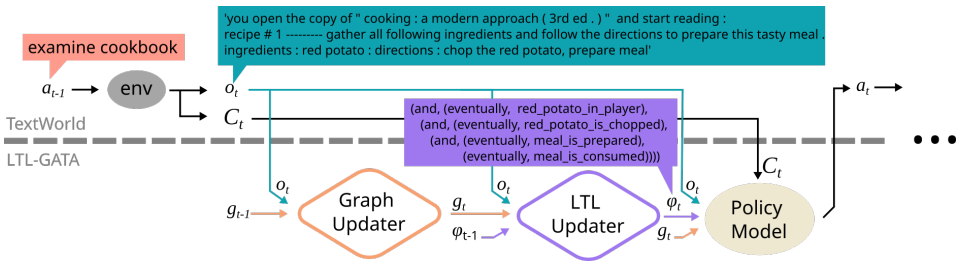


Figure 2: An example of a single step in an episode of TextWorld. The game environment returns an observation o_t and action candidate set C_t in response to action a_{t-1}. In turn, the agent's graph updater (GATA) updates its belief graph g_t in response to both o_t and g_{t-1}. Next, g_t and o_t update the LTL instructions. φ_t is generated from o_t after the cookbook is examined and thereafter φ_{t-1} is progressed to φ_t at each time step. The policy network selects action a_t from C_t conditioned on o_t, φ_t, and g_t and the cycle repeats.

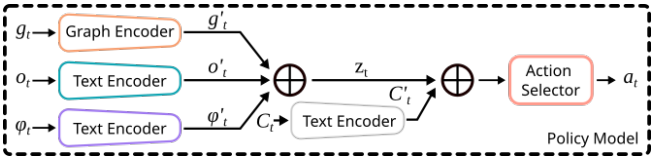


Figure 3: LTL-GATA's policy model. The model chooses action $a_t \in C_t$ conditioned on the state $z_t = [o'_t; \varphi'_t; g'_t]$. The action selector chooses a_t based on the predicted Q-values.

Table 1: Cooking Levels

Level	Recipe Size	Rooms	Max Score	Need {Grab, Cut, Cook}
0	1	1	3	{✓, ✗, ✗}
1	1	1	4	{✓, ✓, ✗}
2	1	1	5	{✓, ✓, ✓}
3	1	9	3	{✓, ✗, ✗}

LTL Updater: The LTL updater generates and progresses LTL instructions. LTL instructions defining the need to arrive at the kitchen and examine the cookbook are generated from the initial observation o_0. The subsequent instruction defining the recipe is generated from game observation o_t, as described in subsection 4.1, when the action *examine cookbook* is executed at time t. For the truth assignments (i.e. the labelling function L), we leverage GATA's highly accurate belief state from the graph updater. We use the Spot engine (Duret-Lutz et al., 2016) to perform the progression.

Text Encoders: For encoding the action choices C_t, observations o_t, as well as encoding the LTL instructions φ_t, we use a simplified version of the Transformer architecture presented in Vaswani et al. (2017). This is the same architecture used in (Adhikari et al., 2020). For LTL instructions, we encode them directly as a string. For example, the LTL formula $\varphi : (\texttt{EVENTUALLY}\, p_1) \wedge (\texttt{EVENTUALLY}\, p_2)$ where $p_1 = \texttt{pepper-in-player}$ and $p_2 = \texttt{pepper-is-cut}$, has the string representation

$$\text{str}(\varphi) : \text{“eventually player_has_pepper and eventually pepper_is_cut“}$$

We format each predicate as a single token, and we show in subsubsection H.4.1 that our method is robust to predicate format. For some input string $v \in \mathbb{R}^{\ell}$ of length ℓ, the text encoder outputs a single vector $\texttt{TE}(v) = v' \in \mathbb{R}^D$ of dimension D, which is the same latent dimension as the graph encoder.

Action Selector: The action selector is a 2-layer multi-layer perceptron (MLP). The encoded state vectors $\texttt{TE}(o_t) = o'_t \in \mathbb{R}^D$, $\texttt{TE}(\varphi_t) = \varphi'_t \in \mathbb{R}^D$, and $\texttt{GE}(g_t) = g'_t \in \mathbb{R}^D$ are concatenated to form the agent's final state representation $z_t = [o'_t; \varphi'_t; g'_t] \in \mathbb{R}^{3D}$. In contrast to Adhikari et al. (2020), we concatenate features rather than use the bi-directional attention-based aggregator. This simplified the model's complexity and worked just as well experimentally. This vector is then repeated n_c times and concatenated with the encoded action choices $C'_t \in \mathbb{R}^{n_c \times D}$ where n_c is the number of action choices. This final input matrix is fed to the MLP which returns a vector $q_c \in \mathbb{R}^{n_c}$, which represents the Q-values for each action choice.

Training. Formally, for belief state g and LTL instruction φ, LTL-GATA aims to learn an optimal policy $\pi^*(a|o, g, \varphi)$. To learn this optimal policy, we implement Double DQN (DDQN) Van Hasselt

et al. (2016) with reward function and termination criteria as discussed in subsection 4.2. We use a prioritized experience replay buffer Schaul et al. (2016). Refer to subsection G.1 for further details.

5 Experiments

Our experimental assessment was designed both to understand how well GATA was exploiting observational instructions, as discussed in section 3, and to assess the instruction-following performance of our proposed approach relative to this state of the art (not only in terms of game points but also successful completion). We additionally strove to assess features of our approach (such as monitoring instruction progress) that contributed to its performance, as well as general challenges to text-based game playing that limited its performance (such as navigation).

5.1 Experimental Setup

Games. Consistent with Adhikari et al. (2020), we use the same set of games in the Cooking domain[1], which has distinct training, validation, and testing sets. For the training games, there are two sets: one set that contains 20 unique games and another that contains 100 unique games. Both the validation and testing sets are set at 20 unique games each. The chosen levels are shown in Table 1.

Hyper-parameters. To remove any bias towards more finely tuned experimental configurations, we replicate all but two hyper-parameters from the original GATA work: (1) we use a batch size of 200 instead of 64 when training on the 100 game set, (2) for level 3, we use Boltzmann Action selection. These changes boosted performance for all models. See subsection H.1 for more details.

Baselines. We compare against (1) TDQN Adhikari et al. (2020), the transformer variant of the LSTM-DQN Narasimhan et al. (2015) model, (2) GATA^{C}, and (3) GATA^{D}. GATA^{C} is GATA's best performing model (GATA-COC) that uses a continue graph-updater pre-trained using constrastive observation classification. GATA^{D} is a similarly performant model (GATA-GTP) that uses a discrete graph-updater pre-trained with ground-truth graphs from the FTWP dataset. Finally, we note that we found a few issues with GATA's original code[2] and have since fixed them (see subsubsection H.5.1). For comparison, we include the original *paper* GATA models, labelled as $\text{GATA}^{\text{C}}_{\text{P}}$ and $\text{GATA}^{\text{D}}_{\text{P}}$.

Measuring performance. We measure performance using two metrics: normalized accumulated game points and game success rate. We report averaged results over 3 seeds for each experiment. Previous works only compared using the normalized accumulated game points, however this may sometimes be misleading — an agent could get $3/4 = 0.75$ points on all games but never actually succeed on any. In contrast, measuring the success rate alongside the normalized game points allows for a more complete analysis of the agent's ability to play and complete these games.

5.2 LTL-GATA Compared to Baselines

Consistently high performance with 20 training games. We see from Figure 4 that LTL-GATA exhibits consistently high performance across levels as compared to the baselines when trained on the 20 games set. In particular, LTL-GATA maintains its performance on level 2, where the game's slight increase in complexity causes large performance drop-offs in other methods. Our agent can easily complete the added task and maintain similar performance to the previous level 1.

Large performance gains with 100 training games. We see from Figure 4 that LTL-GATA gains considerable performance when trained on 100 games. With the added games, our agent is exposed to more predicates and can now generalize better to the testing set. Future work may look at how to achieve this kind of generalization without having to expose our agent to more predicates.

Success rate and normalized game points. Looking at the performance of GATA on level 2, it becomes apparent why measuring the success is important. Although it achieves almost 0.4

[1] https://aka.ms/twkg/rl.0.2.zip, accessed from https://github.com/xingdi-eric-yuan/GATA-public.

[2] https://github.com/xingdi-eric-yuan/GATA-public, released under the open-source MIT License.

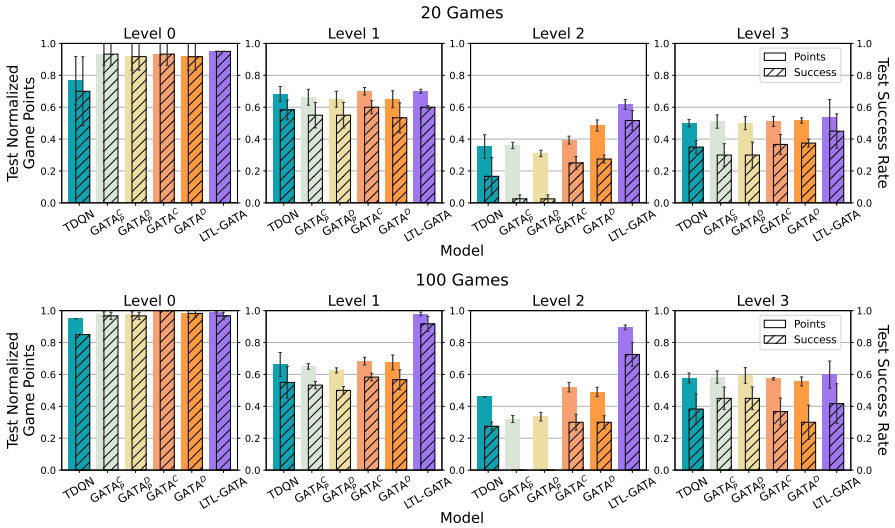
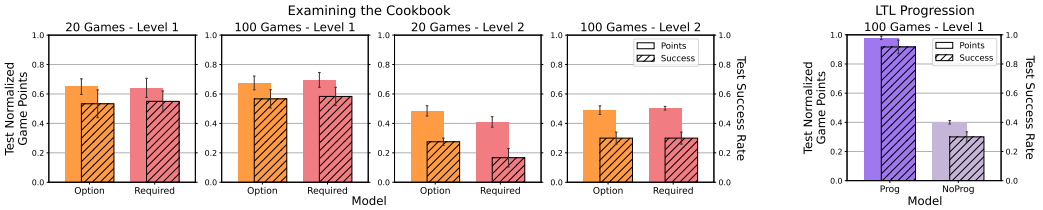


Figure 4: Testing scores across various levels and on both the 20 (top) and 100 (bottom) game training sets. We select the top-performing models (per seed) on the validation set during training and apply those models on the test set and report the average scores.



(a) Letting GATA Consume the Cookbook

(b) Progression Ablation

Figure 5: (a) A comparison of GATA^{D} performance when given the *Option* to examine the cookbook vs. when it is *Required* to examine the cookbook. (b) A comparison of LTL-GATA with (*Prog*) and without (*NoProg*) using LTL progression.

normalized points, the actual success rate is near 0 for original GATA models, and $\sim 60\%$ of the normalized points for the fixed models average across both training sets. In contrast, LTL-GATA exhibits high normalized points and success rate, where the average success rate across both training sets is $\sim 82\%$ of the normalized points.

Competitive performance on level 3. Level 3 introduces the added challenge of navigation. LTL-GATA outperforms GATA in this level as well, but not to the degree of previous levels. Inspecting testing trajectories it becomes evident that both LTL-GATA and GATA methods struggle with navigation in this level, and have difficulties even navigating to the kitchen in the first place.

5.3 Does LTL Progression Matter?

We show in Figure 5b that the use of progression is critical to performance, where LTL-GATA without progression incurs a large performance drop-off, dropping below the performance of the baselines as well. Without progression, the LTL instruction will not reflect the changes incurred by the agent's actions. This appears to confuse the agent considerably, demonstrated by its performance drop-off.

5.4 Letting GATA Consume the Cookbook

Because LTL-GATA is always tasked with examining the cookbook, we question whether a similar tasking for GATA improves performance. We experiment with GATA^{D} by forcing the agent to

examine the cookbook on the first step of the episode. Forcing GATA to examine the cookbook will elicit goal relations like `(apple,needs,cut)` in the belief state. We show however in Figure 5a that GATA does not improve when being given the cookbook. This shows that GATA cannot make use of the information elicited from the cookbook, continuing to ignore important instructions. Even with the presence of goal relations in its belief state, GATA fails to properly attend to this information. This highlights the benefits of a formalized representation of instructions used by LTL-GATA.

6 Related Work

Text-based games. In this work we equip a text-based deep RL agent with formalized LTL instructions, building on previous works that employed belief graphs for solving text-based games. Adhikari et al. (2020) focused on supervised (i.e. translation) and self-supervised learned mechanisms to construct such belief graphs, whereas Ammanabrolu & Hausknecht (2020); Yin & May (2019b); Ammanabrolu & Riedl (2019) employed rule-based methods. At a larger scope, there is a host of other works on playing text-based games using deep reinforcement learning (Hausknecht et al., 2020; Zahavy et al., 2018; Jain et al., 2020; Yin & May, 2019a). Yuan et al. (2018) used count-based memory to shape the reward to improve in exploration and generalization in a simple domain. Narasimhan et al. (2015); He et al. (2016) proposed variations of an LSTM-based model, which the TDQN model used in this work is built from. In just published work, Liu et al. (2022) took a model-based approach, focusing on object-oriented dynamics. However, these works do not address the role and representation of instructions that defines our work. Kimura et al. (2021) does employ a neuro-symbolic RL method using Logical Neural Networks. However, it does not focus on instructions, operates over all logical facts of the environment, and is applied to a simpler domain.

Instruction following and Linear Temporal Logic. Vaezipoor et al. (2021) trained an RL agent to follow various LTL instructions in both discrete and continuous action-space visual environments. They used R-GCNs to learn representations of the LTL instructions and employed LTL progression. Their model showed good generalization performance on similar and much larger unseen instructions than those observed during training . However, they relied on ground-truth event detectors, while we use a learned event detector (GATA) and opt for training the LTL semantics end-to-end using a transformer rather than an R-GCN. Works using LTL for reward specification (Leon et al., 2020; Kuo et al., 2020; Camacho et al., 2019; Toro Icarte et al., 2018b; Littman et al., 2017) or advice (Toro Icarte et al., 2018a) in RL agents exist, however they do not focus on text-based environments nor partially observable ones. Additionally, our work differs in that we model instructions dynamically on the environment's observations and use a learned event detector (GATA) rather than ground truth.

7 Conclusion

We studied the ability of RL agents to follow instructions in text-based games using TextWorld. We conducted experiments to show how current state-of-the-art model-free agents largely fail to exploit instructions and do not typically complete prescribed tasks, and how LTL can be used to construct internal structured representations for state augmentation that result in large performance improvements and more reliable instruction following and task completion. Experiments showed that monitoring instruction progress was critical to these gains. Our method inherits limitations in dealing with navigation and unseen games from prior work, but these concerns are somewhat orthogonal to our focus on instruction following. These present important complementary directions for future work. Finally, we consider the broader impact of this work by relating to the critical need for good instruction following in safety-oriented domains such as autonomous transport or health care. Overall, we intend this paper to highlight the importance of studying instruction following in environments like TextWorld that act as a proxies to the general class of problems dealing with language understanding and human-machine interaction.

References

Adhikari, A., Yuan, X., Côté, M.-A., Zelinka, M., Rondeau, M.-A., Laroche, R., Poupart, P., Tang, J., Trischler, A., and Hamilton, W. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems (NeurIPS2020)*, 33:3045–3057, 2020.

Ammanabrolu, P. and Hausknecht, M. J. Graph constrained reinforcement learning for natural language action spaces. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=B1x6w0EtwH`.

Ammanabrolu, P. and Riedl, M. O. Playing text-adventure games with graph-based deep reinforcement learning. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 3557–3565. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1358. URL `https://doi.org/10.18653/v1/n19-1358`.

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

Bacchus, F. and Kabanza, F. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

Baier, C. and Katoen, J. *Principles of Model Checking*. MIT Press, 2008.

Camacho, A., Toro Icarte, R., Klassen, T. Q., Valenzano, R. A., and McIlraith, S. A. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6065–6073, 2019.

Côté, M., Kádár, Á., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M. J., Asri, L. E., Adada, M., Tay, W., and Trischler, A. Textworld: A learning environment for text-based games. In Cazenave, T., Saffidine, A., and Sturtevant, N. R. (eds.), *Computer Games - 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers*, volume 1017 of *Communications in Computer and Information Science*, pp. 41–75. Springer, 2018. doi: 10.1007/978-3-030-24337-1_3. URL `https://doi.org/10.1007/978-3-030-24337-1_3`.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., and Xu, L. Spot 2.0—A Framework for LTL and ω-Automata Manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pp. 122–129. Springer, 2016.

Hausknecht, M., Ammanabrolu, P., Côté, M.-A., and Yuan, X. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7903–7910, 2020.

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. doi: 10.18653/v1/p16-1153. URL `https://doi.org/10.18653/v1/p16-1153`.

Jain, V., Fedus, W., Larochelle, H., Precup, D., and Bellemare, M. G. Algorithmic improvements for deep reinforcement learning applied to interactive fiction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4328–4336, 2020.

Kimura, D., Ono, M., Chaudhury, S., Kohita, R., Wachi, A., Agravante, D. J., Tatsubori, M., Munawar, A., and Gray, A. Neuro-symbolic reinforcement learning with first-order logic. In Moens, M., Huang, X., Specia, L., and Yih, S. W. (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 3505–3511. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.283. URL https://doi.org/10.18653/v1/2021.emnlp-main.283.

Kuo, Y., Katz, B., and Barbu, A. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, pp. 5604–5610. IEEE, 2020. doi: 10.1109/IROS45743.2020.9341325. URL https://doi.org/10.1109/IROS45743.2020.9341325.

Leon, B. G., Shanahan, M., and Belardinelli, F. Systematic Generalisation through Task Temporal Logic and Deep Reinforcement Learning. *arXiv preprint arXiv:2006.08767*, 2020.

Littman, M. L., Topcu, U., Fu, J., Jr., C. L. I., Wen, M., and MacGlashan, J. Environment-independent task specifications via GLTL. *CoRR*, abs/1704.04341, 2017. URL http://arxiv.org/abs/1704.04341.

Liu, G., Adhikari, A., Farahmand, A.-m., and Poupart, P. Learning object-oriented dynamics for planning from text. In *ICLR 2022*, April 2022. URL https://www.microsoft.com/en-us/research/publication/learning-object-oriented-dynamics-for-planning-from-text/.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

Narasimhan, K., Kulkarni, T. D., and Barzilay, R. Language understanding for text-based games using deep reinforcement learning. In Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., and Marton, Y. (eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 1–11. The Association for Computational Linguistics, 2015. doi: 10.18653/v1/d15-1001. URL https://doi.org/10.18653/v1/d15-1001.

Pnueli, A. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57. IEEE, 1977.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05952.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Advice-based Exploration in Model-based Reinforcement Learning. In *Proceedings of the 31st Canadian Conference on Artificial Intelligence (CCAI)*, pp. 72–83. Springer, 2018a.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461, 2018b.

Trischler, A., Côté, M.-A., and Lima, P. First TextWorld Problems, the competition: Using text-based games to advance capabilities of AI agents. *Microsoft Research Blog*. URL https://www.microsoft.com/en-us/research/blog/first-textworld-problems-the-competition-using-text-based-games-to-advance-capabilities-of-ai-agents/, 2019.

Vaezipoor, P., Li, A. C., Toro Icarte, R. A., and Mcilraith, S. A. LTL2action: Generalizing LTL instructions for multi-task RL. In *International Conference on Machine Learning*, pp. 10497–10508. PMLR, 2021.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yao, S., Narasimhan, K., and Hausknecht, M. J. Reading and acting while blindfolded: The need for semantics in text game agents. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tür, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y. (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pp. 3097–3102. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.naacl-main.247. URL `https://doi.org/10.18653/v1/2021.naacl-main.247`.

Yin, X. and May, J. Comprehensible context-driven text game playing. In *2019 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2019a.

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

Yuan, X., Côté, M., Sordoni, A., Laroche, R., des Combes, R. T., Hausknecht, M. J., and Trischler, A. Counting to explore and generalize in text-based games. *CoRR*, abs/1806.11525, 2018. URL `http://arxiv.org/abs/1806.11525`.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. Learn what not to learn: Action elimination with deep reinforcement learning. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 3566–3577, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/645098b086d2f9e1e0e939c27f9f2d6f-Abstract.html`.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] **Limitations are described in both Section 5 (experiments) and Section 7 (conclusion).**
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] **A broader impact is considered in Section 7 (conclusion), and further discussion can be found in Section I in the supplemental material.**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] **The supplemental material contains all code, and instructions for each experiment in the paper.**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] **Hyper-parameters parameters are discussed briefly in Section 5 and in full detail in Section G and Section H.1 in the supplemental material.**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] **Yes, error bars can be found in every results figure.**
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] **This information can be found in Section H.3 in the supplemental material.**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] **The footnotes in Section 5 include links to the original assets, and in-text citations give credit throughout the paper.**
   (b) Did you mention the license of the assets? [Yes] **The footnotes in Section 5 include links to the original assets.**
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] **The only new asset in this work is our code, which we provide in the supplemental material.**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]