Learning to Follow Instructions in Text-Based Games

Mathieu Tuli, Andrew C. Li, Pashootan Vaezipoor, Toryn Q. Klassen†,
Scott Sanner, Sheila A. McIlraith†
University of Toronto, Toronto, Canada
Vector Institute for Artificial Intelligence, Toronto, Canada
† Schwartz Reisman Institute for Technology and Society, Toronto, Canada
`{mathieutuli,andrewli,pashootan,toryn,sheila}@cs.toronto.edu`
`ssanner@mie.utoronto.ca`

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 with 500+ games in TextWorld 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,

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

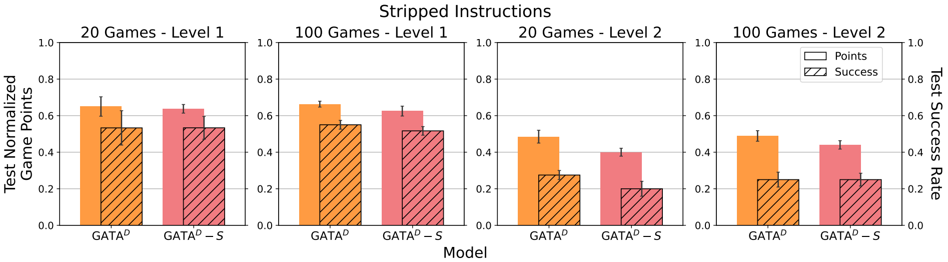


Figure 1: Comparison of GATA performance when trained with instructions (GATA^{D}) versus when instructions are stripped from environment observations (GATA^{D}-S). Agents were trained with 20 or 100 games, at increasing levels of task difficulty (level 1 vs level 2). Note that normalized game point 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 and other purposes in RL (e.g., Bacchus & Kabanza, 2000; Baier & McIlraith, 2006; Patrizi et al., 2011; Camacho & McIlraith, 2019; Littman et al., 2017; Toro Icarte et al., 2018a,b; Camacho et al., 2019; Leon et al., 2020; Kuo et al., 2020; Vaezipoor et al., 2021). 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, $R : S \times A \to \mathbb{R}$ 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 follows.

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 instruction 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 $p_1, p_2, \dots p_n$ be true, the cookbook instructions are modelled as $\varphi : (\texttt{EVENTUALLY}\, p_1) \wedge (\texttt{EVENTUALLY}\, p_2) \wedge \dots (\texttt{EVENTUALLY}\, p_n)$.

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 third and 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.

One possible criticism with such an LTL translator is its reliance on domain knowledge. While not the main focus of this paper, a complementary research problem that has begun to be explored is to *automatically* translate natural language instructions to LTL (e.g., Dzifcak et al., 2009; Finucane et al., 2010; Wang et al., 2020). Traditionally, such approaches have required large corpora of training data or hard-coded rules, and were restricted to a specific domain. However, pretrained large language models such as GPT-3 introduce the potential for a general natural-language-to-LTL translation scheme with minimal domain-specific adaptation (Hahn et al., 2022; Huang et al., 2022; Brohan et al., 2022). We explore this prospect by applying GPT-3 to TextWorld in subsection 5.5.

Finally, we note that in this work, GATA provides the domain-dependent vocabulary for describing properties of state (e.g. `carrot-is-chopped`) while our LTL augmentation provides the *domain-independent* temporal modalities (i.e., $\texttt{NEXT}, \texttt{EVENTUALLY}$, etc.) and the logical connectives for composing those properties of state into the instructions we use. In this way, our technique is very generalizable, limited only by the recognizable properties of state (which in our case are provided by GATA) and instructions that can be extracted in game-play.

4.2 LTL Augmented Rewards and Episode Termination

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

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

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.3.2. 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.3.2.

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 Figure 3 depicts the model itself. Additional details can be found in Appendix F.

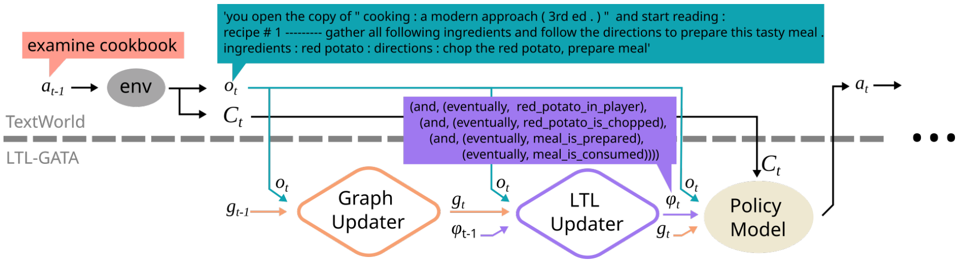

examine cookbook
'you open the copy of " cooking : a modern approach (3rd ed .) " and start reading : recipe # 1 --------- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : red potato : directions : chop the red potato, prepare meal'
a_{t-1}
env
o_t
C_t
TextWorld
LTL-GATA
(and, (eventually, red_potato_in_player), (and, (eventually, red_potato_is_chopped), (and, (eventually, meal_is_prepared), (eventually, meal_is_consumed))))
a_t
g_{t-1}
o_t
Graph Updater
g_t
φ_{t-1}
o_t
LTL Updater
φ_t
g_t
o_t
C_t
Policy Model

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.

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.

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 by Vaswani et al. (2017). This is the same architecture used by 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.3.1 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 actino choices $C'_t \in \mathbb{R}^{n_c \times D}$ where n_c is the number of action choices. This input matrix is fed to the MLP which returns the a vector of Q-values for each action $q_c \in \mathbb{R}^{n_c}$.

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.2 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

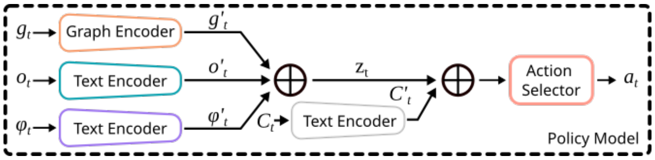


Figure 3: LTL-GATA's policy model. The model chooses action $a_t \in C_t$ conditioned on the state $z_t = [o'_t; \varphi'_t; g'_t]$. The action selector chooses a_t based on the predicted Q-values.

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	{✓, ✗, ✗}

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).[1]

5.1 Experimental Setup

Games. To have as fair a comparison with Adhikari et al. (2020) as possible, we reused the sets of games they had generated. For the training games, they had created two sets: one set that contains 20 unique games per level and another that contains 100 unique games per level. Both the validation and testing sets have 20 unique games each per level. The levels we chose to use in our assessment are shown in Table 1. Note that in our assessment we omit Levels 4 and 5. Level 4 is an augmentation to Level 3 that adds more ingredients; both GATA and LTL-GATA at this level suffer from the navigation issues we discuss later with respect to Level 3. As we wanted to focus on instruction following and not navigation, we omitted this level and chose to use Level 0 instead. Level 5 is simply a random combination of all levels, so it is omitted for similar reasons.

Hyper-parameters. We replicate all but three hyper-parameters from Adhikari et al. (2020): (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, and (3) we use Adam Kingma & Ba (2015) with a learning rate of 0.0003 instead of RAdam Liu et al. (2020) with a learning rate of 0.001. 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 contrastive 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.4.1). For comparison, we include the original *paper* GATA models, labelled as GATA$^{\mathrm{C}}_{\mathrm{P}}$ and GATA$^{\mathrm{D}}_{\mathrm{P}}$.

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.

---

[1] Our code for the experiments can be found at `https://github.com/MathieuTuli/LTL-GATA`

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

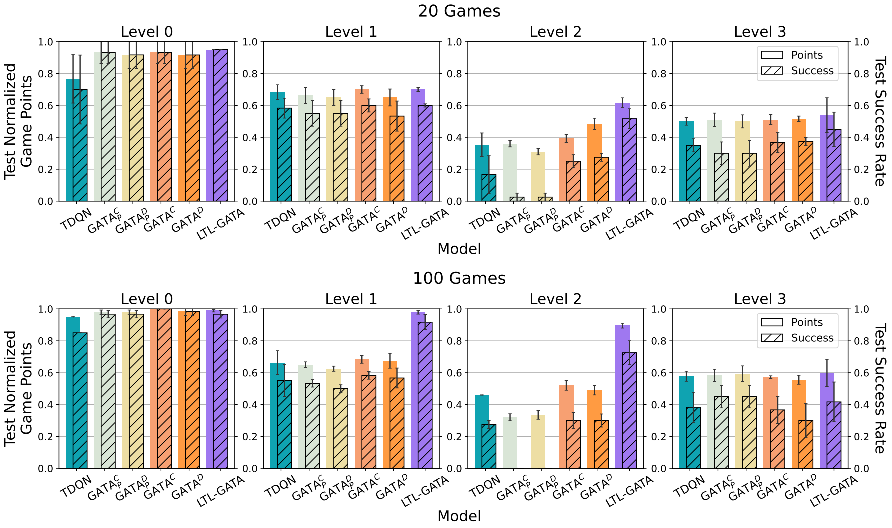
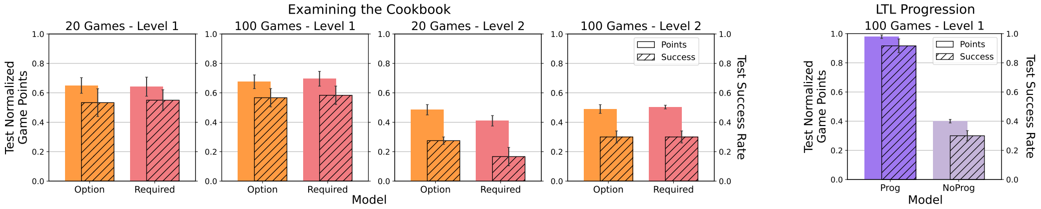


Figure 4: Testing scores across various levels and on both the 20 (top) and 100 (bottom) game training sets. We select the top-performing models (per seed) on the validation set during training and apply those models on the test set and report the average scores.



(a) Forcing GATA to Examine 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.

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 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. Exploring at test time to find items and rooms in an unknown environment is a major challenge built into text-based games. Hypothetically, LTL could contribute to addressing this challenge. LTL could be used to dictate strategy and/or to simply track such exploration (e.g., for remembering which rooms have been previously visited). LTL might also be used to encode *learned* navigation instructions (e.g. “find the blue door, go through it, then turn right”). We do not pursue this vector of research here, but it is an interesting direction for future work.

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 Forcing GATA to Examine 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.

5.5 On Automatic Translation: Natural Language Instructions to LTL

While LTL-GATA relies on a handcrafted LTL translator to provide initial instructions from text observations, we investigate the potential of automating this step using pretrained large language models. This is not a central focus of the paper. Rather, we include this exploration as a proof of concept that the use of LTL is not a barrier to broad deployment of the work presented here. To this end, we evaluate whether GPT-3 (Brown et al., 2020) can few-shot learn to translate TextWorld observations to LTL, given only *six* examples and without additional training.

We experiment with two models of GPT-3 from OpenAI: *Ada* (the fastest model) and *Da Vinci* (the most powerful model). We perform few-shot translation by constructing prompts that contain six example translations, followed by the natural language observation to translate (the test case). The examples remain fixed for all test cases, and follow the form "NL: `<natural language observation>`. LTL: `<ltl-formulas>`". Our test case follows the form "NL: `<natural language observation>`. LTL:", where the model must complete the prompt, thereby performing a translation. We consider a response that exactly matches the ground-truth LTL formula as *absolutely correct*, a response that is otherwise correct except for parentheses and spaces as *almost correct*, and all other responses as *incorrect*. Further details and examples can be found in subsection H.6.

Out of 234 test cases, *Da Vinci* translated 93.2% *absolutely correctly* and another 5.6% *almost correctly*, with only 1.3% of examples incorrect. *Da Vinci* displayed an impressive ability to generalize to unseen adjectives (e.g. `is_grilled`), nouns (e.g. `carrot`), and compositions of formula. Unfortunately, the weaker model, *Ada*, translated all 100% of examples incorrectly. We found that *Ada* commonly hallucinated new nonsensical words and predicates such as `ingredient_is_salt_is_diced` or `banana_pork_chop_in_player`, leading to erroneous translations.

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) and 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 also employed LTL progression. Their model showed good generalization performance on similar and much larger unseen instructions than those observed during training . However, in contrast to the work presented here, they relied on ground-truth event detectors and operated in fully observable settings, while we use GATA's learned belief graphs, in a partially observable setting, to evaluate the truth or falsity of propositions and to progress formulae. We further distinguish ourselves from this work by opting 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.

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. We then showed 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.

Furthermore, we can 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. We would like to suggest that works towards building better language agents should also emphasize the importance of *completing* instructions. To illustrate, for an agent to help a person half-way across a street, or to start but not finish a medical operation, may be worse than for it to do nothing at all. To that end, we have proposed using (game) success rate as a metric for future work, and demonstrated how LTL-GATA is very successful in the games it plays, relative to the state-of-the-art. 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.

Finally, in follow-on work we would like to explore more complex text-based games such as the Jericho environment (Hausknecht et al., 2020). These games involve a number of distinct challenges, including exploration, navigation, puzzle solving, language understanding, and instruction following. In this vein, we'd like to see whether LTL can be exploited to capture (learned) domain-specific strategic advice, or memory to tackle both navigation and exploration challenges. We'd like to further explore seamless ways to exploit the merits of natural language together with the benefits afforded by the compositional syntax and semantics of formal languages such as LTL. To this end, further advancing our explorations translating natural language to LTL is of interest and import, for this and a diversity of other applications in and outside RL.

Acknowledgements

We thank the NeurIPS reviewers for their constructive feedback, and also the reviewers from the *Wordplay: When Language Meets Games* workshop at NAACL 2022, where a preliminary version of this paper appeared (Tuli et al., 2022). We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (`www.vectorinstitute.ai/partners`). Finally, we thank the Schwartz Reisman Institute for Technology and Society for providing a rich multi-disciplinary research environment.

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`.

Ammanabrolu, P., Jiang, L., Sap, M., Hajishirzi, H., and Choi, Y. Aligning to social norms and values in interactive narratives. *CoRR*, abs/2205.01975, 2022. doi: 10.48550/arXiv.2205.01975. URL `https://doi.org/10.48550/arXiv.2205.01975`.

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.

Baier, J. and McIlraith, S. Planning with temporally extended goals using heuristic search. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS06)*, pp. 342–345, June 2006.

Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. Do as I can, not as I say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020. URL `https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Camacho, A. and McIlraith, S. A. Strong fully observable non-deterministic planning with LTL and LTL-f goals. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5523–5531, 2019.

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.

Dzifcak, J., Scheutz, M., Baral, C., and Schermerhorn, P. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4163–4168. IEEE, 2009.

Finucane, C., Jing, G., and Kress-Gazit, H. LTLMoP: Experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1988–1993. IEEE, 2010.

Hahn, C., Schmitt, F., Tillman, J. J., Metzger, N., Siber, J., and Finkbeiner, B. Formal specifications from natural language. *arXiv preprint arXiv:2206.01962*, 2022.

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`.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

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`.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

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/`.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond. 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=rkgz2aEKDr`.

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`.

Patrizi, F., Lipovetzky, N., Giacomo, G. D., and Geffner, H. Computing infinite plans for LTL goals using a classical planner. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2003–2008. IJCAI/AAAI, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-334. URL `https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-334`.

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.

Tuli, M., Li, A., Vaezipoor, P., Klassen, T. Q., Sanner, S., and McIlraith, S. A. Instruction following in text-based games. In *Wordplay: When Language Meets Games Workshop @ NAACL 2022*, 2022.

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. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wang, C., Ross, C., Kuo, Y., Katz, B., and Barbu, A. Learning a natural-language to LTL executable semantic parser for grounded robotics. In Kober, J., Ramos, F., and Tomlin, C. J. (eds.), *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pp. 1706–1718. PMLR, 2020. URL `https://proceedings.mlr.press/v155/wang21g.html`.

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 and section 7.**
    (c) Did you discuss any potential negative societal impacts of your work? [Yes] **A broader impact is considered in section 7, and further discussion can be found in Appendix I.**
    (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] **All code can be found at** `https://github.com/MathieuTuli/LTL-GATA`
    (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 Appendix G and subsection H.1.**
    (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 subsection H.3.**
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 here:** `https://github.com/MathieuTuli/LTL-GATA`**.**
    (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]

Appendices

A Reinforcement Learning **16**

B Partially Observed Reinforcement Learning **16**

C TextWorld: Cooking Domain **17**

D An example of LTL progression **17**

E Generating LTL in TextWorld **18**

F Model **19**

F.1 Text Encoder 19

F.2 Encoder Independence 19

F.3 Action Selector 20

G Implementation Details **20**

G.1 Augmenting GATA's Pre-Training Dataset 20

G.2 Training 20

H Experiments **21**

H.1 Hyper-Parameters 21

H.2 Computational Requirements 21

H.3 Additional Results 21

H.4 Code 23

H.5 Training Curves 23

H.6 Automated Translation: Natural Language Instructions to LTL Details 26

I Broader Impact **28**

A Reinforcement Learning

Reinforcement Learning (RL) is the problem of training machine learning models to solve sequential decision making problems. By interacting with an environment, RL agents must learn optimal behaviours given the current state of their environment. If the environment is fully observable, we can frame it as a Markov Decision Process (MDP) modelled as $\langle S, A, T, R, \gamma \rangle$ 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, $r_t = R(s, a) : S \times A \rightarrow \mathbb{R}$ is the reward function for state action pair (s, a), and $\gamma \in [0, 1]$ is the discount factor. The goal for an RL agent is to learn some optimal policy $\pi^*(a|s)$ that maximizes the expected discounted return $\mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| S_t = s \right]$. A single game is an *episode*, and steps in an episode are indexed by t.

B Partially Observed Reinforcement Learning

In a partially observed environment, an agent does not have access to the full state space S. We can frame this environment as a Partially Observable MDP (POMDP) modelled by $\langle S, A, T, O, \Omega, R, \gamma \rangle$.

In this new setting, $\langle S, A, T, R, \gamma \rangle$ remain unchanged, O represents the set of (partial) observations that the agent receives and $\Omega(o_t|s_t, a_{t-1})$ is the set of conditional observation probabilities. 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.

C TextWorld: Cooking Domain

We present two examples of observations with instructions in Table 2 and highlight where the instructions are and where the rewards come from.

Table 2: TextWorld observations for the Cooking Domain game. We show the observations and highlight where the instructions are, and finally identify what the rewards would be. This is for a level 2 game, and the total possible reward is 5.

Initial Game Observation

"`You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!`" `-= kitchen =- you find yourself in a kitchen. You start to take note of what's in the room. You can make out a closed fridge nearby. You can see an oven. You can make out a table. You wonder idly who left that here. You see a knife on the table. Something scurries by right in the corner of your eye. Probably nothing. You see a counter. The counter is vast. On the counter you see a raw red potato and a cookbook. You see a stove, but the thing is empty, unfortunately.`"

Reward

There is a reward of 1 given for eating the meal. i.e. the instruction "`Once done, enjoy your meal!`" will result in a reward of 1 *after* the recipe has been completed. Note that the instruction "`Check the cookbook in the kitchen for the recipe.`" is not bound to a reward.

Observation following the `examine cookbook` action

"`You open the copy of` *`"Cooking : a modern approach (3rd ed.)"`* `and start reading: recipe #1 ----- Gather all following ingredients and follow the directions to prepare this tasty meal. Ingredients: red potato: directions: chop the red potato, fry the red potato, prepare meal`"

Reward

There are 4 rewards from the instruction "`Gather all following ingredients and follow the directions to prepare this tasty meal. Ingredients: red potato: directions: chop the red potato, fry the red potato, prepare meal`":

- 1 for grabbing the red potato
- 1 for chopping the red potato
- 1 for frying the red potato
- 1 for preparing the meal

D An example of LTL progression

To illustrate how progression works, the LTL instruction $($`EVENTUALLY player-has-carrot`$) \wedge ($`EVENTUALLY player-has-apple`$)$ would be progressed to $($`EVENTUALLY player-has-apple`$)$ once the agent grabs the carrot during an episode. In other words, when the agent reaches a state where `player-has-carrot` is true, the LTL instruction is progressed to reflect that the agent no longer needs to get the carrot but must still grab the apple at some point.

E Generating LTL in TextWorld

We provide some examples of the LTL instructions used in this work in Table 3, Table 4, and Table 5. We build a simple translator that reads game observations and constructs these LTL instructions directly, but only once. Repeated observations will not result in the same LTL formula being generated. Once a formula has been generated, LTL progression is used with the agent's belief state to progress the instructions along the truth assignments: observations are not directly used in the progression, although they do indirectly affect the progression by affecting the belief state.

For levels 0, 1, and 2, the LTL instructions that an agent can receive throughout an episode are (a) the task to examine the cookbook and (b) the recipe-bound task. In other words, the set of un-progressed instructions Φ it can receive over the course of an episode (assuming the cookbook is examined) is as follows:

$$\begin{aligned}\Phi : [\,&\texttt{NEXT}\,\texttt{cookbook-is-examined},\\ &(\texttt{EVENTUALLY}\,p_1) \wedge (\texttt{EVENTUALLY}\,p_2) \wedge \ldots (\texttt{EVENTUALLY}\,p_n)]\end{aligned}$$

where the recipe requires that predicates $p_1, p_2, \ldots p_n$ be true. Note that we also consider eating the meal to be a part of recipe in this case, although it is not explicitly mentioned in the recipe. Further, we note that the "prepare meal" task is represented by the predicate `meal-in-player`, as this is the event that occurs when the meal is prepared in the game.

For levels with navigation (i.e. level 3),

$$\begin{aligned}\Phi : [\,&\texttt{EVENTUALLY}\,\texttt{player-at-kitchen},\\ &\texttt{NEXT}\,\texttt{cookbook-is-examined},\\ &(\texttt{EVENTUALLY}\,p_1) \wedge (\texttt{EVENTUALLY}\,p_2) \wedge \ldots (\texttt{EVENTUALLY}\,p_n)]\end{aligned}$$

where the agent has the added task of first navigating to the kitchen. This instruction provides no help for actually how to arrive at the kitchen, only that the agent must do so. As a result, LTL-GATA still suffers from the difficulties of exploration, and perhaps investigating how LTL can be used to improve in navigation could be a direction for future work.

In total, LTL generation occurs only twice for any level, either during the initial observation or when the cookbook is read. When multiple instructions are generated at once, the agent will process them sequentially, in the order they are given.

Table 3: Level 3 observation and resulting generated LTL instruction

Observation

`“You are hungry! Let’s cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!” -= corridor =- “You’ve entered a corridor. There is a closed screen door leading west. You don’t like doors? Why not try going north, that entranceway is not blocked by one. You need an exit without a door? You should try going south.”`

Generated LTL

This observation will generate two instructions: First,

$$\varphi : (\texttt{EVENTUALLY}\,\texttt{player-at-kitchen})$$

and second,

$$\varphi : (\texttt{NEXT}\,\texttt{cookbook-is-examined})$$

Table 4: Level 1 observation and resulting generated LTL instruction

Observation
"You open the copy of *"Cooking : a modern approach (3rd ed.)"* and start reading: recipe #1 ----- Gather all following ingredients and follow the directions to prepare this tasty meal. Ingredients: red potato: directions: chop the red potato, prepare meal"
Generated LTL
$\varphi : (\texttt{EVENTUALLY}\ \texttt{red-potato-in-player}) \wedge (\texttt{EVENTUALLY}\ \texttt{red-potato-is-chopped}) \wedge (\texttt{EVENTUALLY}\ \texttt{meal-in-player}) \wedge (\texttt{EVENTUALLY}\ \texttt{meal-is-consumed}).$

Table 5: Level 2 observation and resulting generated LTL instruction

Observation
"You open the copy of *"Cooking : a modern approach (3rd ed.)"* and start reading: recipe #1 ----- Gather all following ingredients and follow the directions to prepare this tasty meal. Ingredients: red potato: directions: chop the red potato, fry the red potato, prepare meal"
Generated LTL
$\varphi : (\texttt{EVENTUALLY}\ \texttt{red-potato-in-player}) \wedge (\texttt{EVENTUALLY}\ \texttt{red-potato-is-chopped}) \wedge (\texttt{EVENTUALLY}\ \texttt{red-potato-is-fried}) \wedge (\texttt{EVENTUALLY}\ \texttt{meal-in-player}) \wedge (\texttt{EVENTUALLY}\ \texttt{meal-is-consumed}).$

F Model

F.1 Text Encoder

The text encoder is a simple transformer-based model, with a transformer block (Vaswani et al., 2017) and word embedding layer. We use the pre-trained 300-dimensional fastText (Mikolov et al., 2017) word embeddings, which are trained on Common Crawl (600B tokens). These word embeddings are frozen during training. Strings are tokenized by spaces.

The transformer block is composed of: **(1)** a stack of 5 convolutional layers, **(2)** a single-head self-attention layer, and **(3)** a 2-layer MLP with ReLU non-linear activation function in between. The convolutional layers each have 64 filters, with kernel sizes of 5 and are each followed by a Layer Norm (Ba et al., 2016). We also use standard positional encoding (Vaswani et al., 2017). The self-attention layer uses a hidden size H of 64. The Text Encoder outputs a single feature vector $v \in \mathbb{R}^D$, where $D = 64$ in our experiments.

F.2 Encoder Independence

Figure 3 in the main paper visualizes each component of our model. Specifically, our model has four encoders: **(1)** Graph Encoder, **(2)** Text Encoder for observations, **(3)** Text Encoder for LTL instructions, and **(4)** Text Encoder for action choices. We note here that each of these encoders are independent models, trained concurrently. This is in contrast to the original GATA model that used the same Text Encoder for both the actions and the observations. Because these Text Encoders are relatively small transformers, there is no issues with fitting this model in memory. As shown in Table 6, the model is still quite efficient, even more than the original GATA code. We found that using independent encoders resulted in better performance than using a single Text Encoder that would have been responsible for encoding the observations, LTL instructions, and action choices.

F.3 Action Selector

The action selector is a simple two-layer MLP with a ReLU non-linear activation function in between. It takes as input, at time step t, the concatenated representation of the agent's state vector $z_t \in \mathbb{R}^{3D}$ and the action choices $C'_t \in \mathbb{R}^{n_c \times D}$. Recall that in our experiments $D = 64$. The first layer uses an input dimension of $4D$ and an output dimension of D. The second layer has an input dimension of D and output dimension of 1, which after squeezing the last dimension during the forward pass, the final output vector $q_c \in \mathbb{R}^{n_c}$ represents the q-values for each action choice.

The input to the action selector is constructed by repeating the agent's state representation, z_t, n_c times and then concatenating with the encoded actions choices C'_t. We wanted to further explain why this occurs, as it may not be immediately clear. The action selector in this work is a parameter-tied Q-value predictor. That is, for some action $a_i \in C_t,\ i \in [1, \ldots, n_c]$ and agent state representation z_t, the predicted Q-value is $q_i = \texttt{AS}([a_i, z_t])$. Thus, the action selector (i.e. $\texttt{AS}(\cdot)$) predicts Q-values given action a_i and agent state representation z_t. Thus, during a single episode step, given our encoded actions choices $C'_t \in \mathbb{R}^{n_c \times D}$, in order for the action selector to predict Q-values for each of these action choices, we repeat $z_t \in \mathbb{R}^{3D}$ n_c times and stack it together, which results in a state matrix $Z_t \in \mathbb{R}^{n_c \times 3D}$. When we concatenate this matrix with our action choices we are left with the input to our action selector: $[C'_t; Z_t] \in \mathbb{R}^{n_c \times 4D}$. Looking at this matrix, each row in this input matrix is effectively the concatenation of action a_i with agent state representation z_t, and so passing this matrix to our action selector performs the parameter-tied Q-value prediction $q_i = \texttt{AS}([a_i, z_t])$ for all action choices, and outputs a single vector of Q-values for each action $q_c \in \mathbb{R}^{n_c}$. We can then use these predicted Q-values to perform action selection using either a greedy approach, an ϵ-greedy approach, Boltzmann action selection, etc.

G Implementation Details

G.1 Augmenting GATA's Pre-Training Dataset

We note here that although possible, the vocabulary and dataset used by Adhikari et al. (2020) did not allow for the knowledge triple $\{cookbook, is, examined\}$ to be extracted from observations. Without this triple being extracted and added to the agent's belief state, there would be no way for the agent to progress LTL instructions requiring the agent to examine the cookbook. In our pre-training of the GATA graph encoder, we augmented the dataset provided by Adhikari et al. (2020) to include the triplet $\{cookbook, is, examined\}$ when relevant (i.e. when the agent examines the cookbook). This was a simple process of adding this triple to the ground truth belief graphs in the dataset so that during pre-training, GATA could learn how to translate these triplets from relevant observations.

G.2 Training

For training to learn our optimal policy we use the Double-DQN (DDQN) (Van Hasselt et al., 2016) framework. We use ϵ-greedy for training, which first starts with a warm-up period, using a completely random policy (i.e. $\epsilon = 1.0$) for the first $1,000$ episodes. We then anneal ϵ from 1.0 to 0.1 over the next $3,000$ episodes after the initial warm-up (i.e. episodes $1,000$ to $4,000$). We use a prioritized experience replay buffer ($\alpha = 0.6$ and $\beta = 0.4$) with capacity $500,000$. For DDQN, the target network updates occur every 500 episodes. We update network parameters every 50 game steps, and we play 50 games in parallel.

We train all agents for $100,000$ episodes using a discount factor of 0.9, and we use $\{123, 321, 666\}$ as our random seeds. Each episode during training is limited to a maximum of 50 steps, and during testing/validation this limit is increased to 100 steps. We report results and save checkpoints every $1,000$ episodes. We also use a patience window p that reloads from the previous best checkpoint during training when validation performance has decreased for p episodes in a row. This is the same strategy used in Adhikari et al. (2020). For our experiments, we used $p = 3$.

For reporting testing results, each model is trained using the three seeds mentioned before, and fine-tuned on the validation set. That is, the checkpoint of the model that performs best on the validation set during training is saved, and each of these models (three, one for each seed) is applied to the test set. Reported test results are the average over these three models.

H Experiments

H.1 Hyper-Parameters

To have as fair a comparison as possible, we replicate all but three hyper-parameters from the settings used in Adhikari et al. (2020). We do this to remove any bias towards more finely tuned experimental configurations and focus only on the LTL integration. Further, we re-run the GATA experiments to confirm their original results. The three changes we implemented were (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, and (3) we use Adam (Kingma & Ba, 2015) with a learning rate of 0.0003 instead of RAdam (Liu et al., 2020) with a learning rate of 0.001. These changes boosted performance for all models. For the 20 training game set, we use a batch size of 64.

For Boltzmann action selection, we used a temperature of $\tau = 100$. We experimented with various temperatures ($\tau \in \{1, 10, 25, 50, 100, 200\}$) and found $\tau = 100$ to perform the best across models.

H.2 Computational Requirements

We report the wall-clock times for our experiments in Table 6.

Table 6: Training times for each model and training set size. The times were reported using a workstation with dual RTX3090s, an AMD Ryzen 5950x 16-core CPU, and 128GB of RAM. For the graph updater, COC stands for the contrastive observation classification pre-training (the continuous belief graph model) and GTP stands for ground-truth pre-training (the discrete belief graph model).

Model	Training Set Size	Batch Size	Approximate Time
TDQN	20	64	16 hours
LTL-GATA	20	64	24 hours
GATA^{D}	20	64	24 hours
GATA^{C}	20	64	24 hours
GATA$_{\text{P}}^{\text{D}}$	20	64	36 hours
GATA$_{\text{P}}^{\text{C}}$	20	64	36 hours
TDQN	100	200	32 hours
LTL-GATA	100	200	48 hours
GATA$^{\text{D}*}$	100	200	48 hours
GATA^{C}	100	200	48 hours
GATA$_{\text{P}}^{\text{D}}$	100	200	65 hours
GATA$_{\text{P}}^{\text{C}}$	100	200	65 hours
Graph Updater using COC	N/A	64	48 hours
Graph Updater using GTP	N/A	64	48 hours

H.3 Additional Results

H.3.1 Ablation: Formatting LTL Predicates

As we saw from Figure 4, LTL-GATA when trained on the 100 games set performs significantly better than when trained on the 20 game set, which we attribute to the increased exposure to predicates during training, allowing it to generalize better during testing. To see if we can achieve the same level of generalization when training on the 20 game set, we compare LTL-GATA with LTL predicates represented as single tokens (what we did in the main paper) with using multiple tokens. That is, we compare the following two string representations:

(single-token predicates) str(φ) : “eventually player_has_pepper and eventually pepper_is_cut”
(multi-token predicates) str(φ) : “eventually player has pepper and eventually pepper is cut”

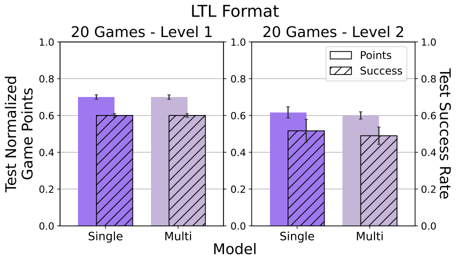
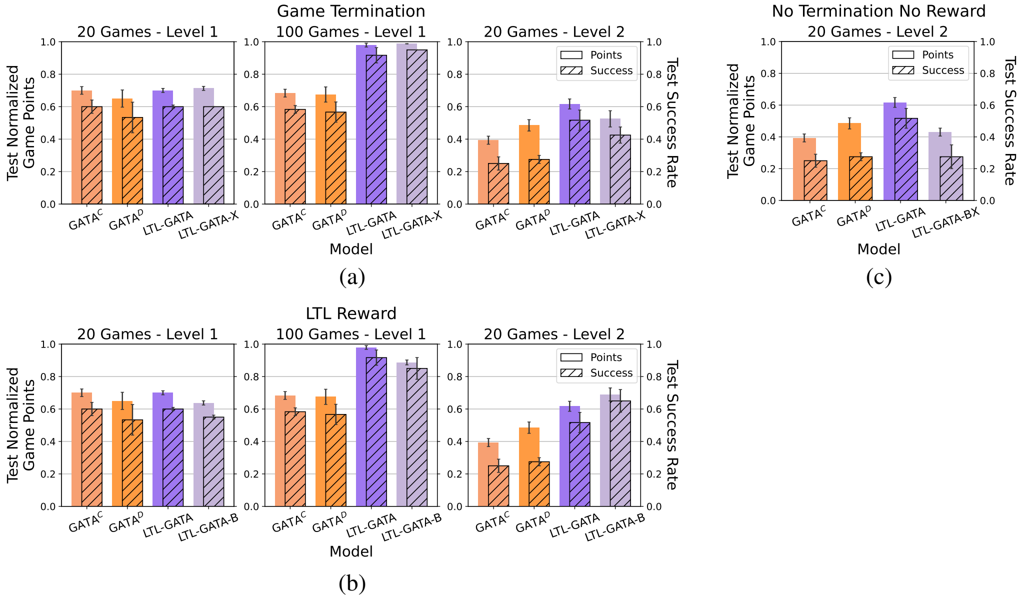


Figure 6: Study on LTL predicate format with single-token (*Single*) predicates and multi-token (*Multi*) predicates. Performance is largely unchanged with predicate format.



(a)

(c)

(b)

Figure 7: Ablation studies on LTL-based episode termination and new reward function $R_{\text{LTL}}(s, a, \varphi)$ and on LTL progression. (a) LTL-GATA with new reward function $R_{\text{LTL}}(s, a, \varphi)$ and without LTL-based episode termination (*LTL-GATA-X*). (b) LTL-GATA with base game reward function $R(s, a)$ and with LTL-based episode termination (*LTL-GATA-B*). (c) LTL-GATA with base game reward function $R(s, a)$ and without LTL-based episode termination (*LTL-GATA-BX*).

The single-token predicates are mapped in the vocabulary to a single word embedding. In our work, we compute word embedding for these single-token predicates by averaging the word embeddings of each underscore-separated word in the predicate. For example, the word embedding (`WE`) of the token `player_has_pepper` is

$$\texttt{WE}(\texttt{player\_has\_pepper}) = \frac{\texttt{WE}(\texttt{player}) + \texttt{WE}(\texttt{has}) + \texttt{WE}(\texttt{pepper})}{3}$$

For multiple-token predicates, each word has its own word embedding and we treat each word as any other word in the sentence. The idea is that by separating the tokens in the predicates, the text encoder (transformer) may be able to attend to each token independently, and during testing have better generalization. We visualize the results of this study in Figure 6. We can see from Figure 6 that this in fact does not help, and LTL-GATA performs almost equally in either scenario. This does however show how our method is robust to predicate format.

H.3.2 The Effect of LTL Reward and LTL-Based Termination

It is important to study the effect that the additional LTL bonus reward and LTL-based episode termination has on the performance of LTL-GATA. To study this, we consider three scenarios: **(a)**

LTL-GATA with the new reward function $R_{\text{LTL}}(s, a, \varphi)$ and without LTL-based episode termination; **(b)** LTL-GATA with the base TextWorld reward function $R(s, a)$ and LTL-based episode termination; and **(c)** LTL-GATA with the normal TextWorld reward function $R(s, a)$ and without LTL-based episode termination. For (a) and (b) we select level 1 on both the 20 and 100 game training set and level 2 on the 20 game training set. For (c) we select level 2 on 20 training games. We visualize the ablative study of these three scenarios in Figure 7.

From Figure 7 we can conclude that the presence of either the new reward function $R_{\text{LTL}}(s, a, \varphi)$ *or* LTL-based episode termination is important to the performance of LTL-GATA. This is because either of these methods will incentivize the agent to complete the initial `NEXT cookbook-is-examined` instruction, which isn't intrinsically rewarded by TextWorld. We can demonstrate the importance of this incentive by analyzing just one level (level 2 on 20 training games). Removing both methods leads to the agent not examining the cookbook, preventing it from receiving further instructions, which we can see from Figure 7(c) results in considerable performance loss, regressing to the baseline GATA.

H.4 Code

All code for this work can be found at `https://github.com/MathieuTuli/LTL-GATA`.

H.4.1 Fixing the GATA code

We found two primary issues in the GATA code. First, we noticed that their implementation of the double Q-learning error was wrong. For Double Q-Learning, after performing some action a_t in state s_t and observing the immediate reward r_t and resulting state s_{t+1}, the Q-Learning error is defined per Van Hasselt et al. (2016) as

$$Y_t = r_t + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t) \tag{1}$$

where $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}'_t$ are the parameters of the policy network and the target network, respectively. However, we noticed that the original code for GATA was computing the error as [3]

$$Y_t = r_t + r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

In other words, the reward for the stepped state was also being added to the error.

Second, we found that the double Q-learning error for terminal states was being incorrectly implemented. Specifically, when computing the error for the case where s_t is a terminal state, and therefore the stepped state s_{t+1} does not exist, the stepped state was not being masked [4]. Additionally, presumably because of this initial error, terminal states were very rarely returned when sampling from experience, unless certain criteria were met [5].

We found fixing these issues improved GATA's performance considerably, which we demonstrated in Figure 4, and all our experimental results for GATA have this correction implemented.

H.5 Training Curves

Here we present accompanying training curves for experiments reported in this work. We report averaged curves of the normalized accumulated reward with bands representing the standard deviation.

---

[3] `https://github.com/xingdi-eric-yuan/GATA-public/blob/c1afc3c9ab38256f839b3e0ddf8243796df5bd77/dqn_memory_priortized_replay_buffer.py#L120-L123`

[4] `https://github.com/xingdi-eric-yuan/GATA-public/blob/c1afc3c9ab38256f839b3e0ddf8243796df5bd77/agent.py#L1353-L1369`

[5] `https://github.com/xingdi-eric-yuan/GATA-public/blob/c1afc3c9ab38256f839b3e0ddf8243796df5bd77/dqn_memory_priortized_replay_buffer.py#L93-L102`

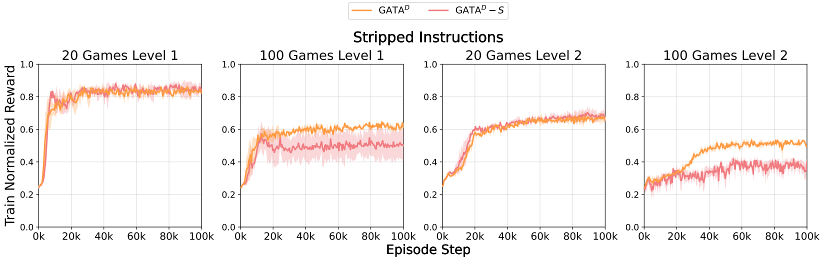


Figure 8: Training curves (of normalized accumulated reward) for the comparison of GATA when trained with instructions (GATA$^\text{D}$) versus when instructions are stripped from environment observations (GATA$^\text{D}$-S). Agents were trained with 20 or 100 games, at increasing levels of task difficulty (level 1 vs level 2). Bands represent the standard deviation.

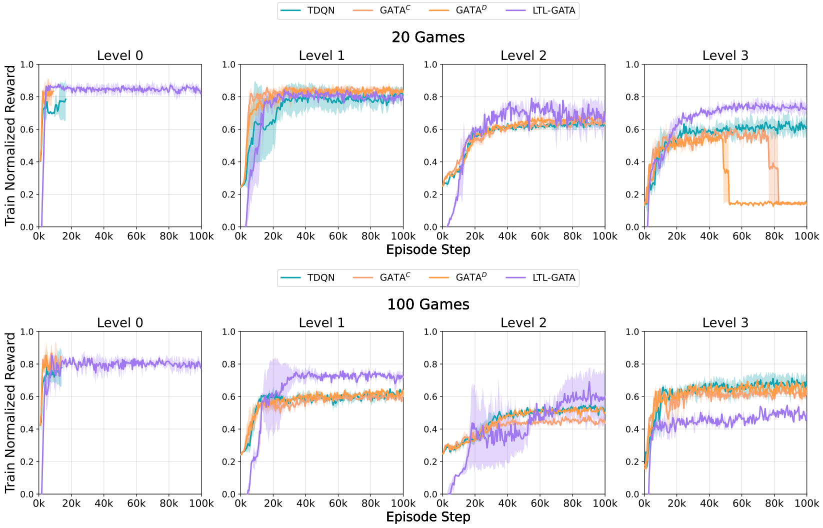


Figure 9: Training curves of the normalized accumulated reward across various levels and on both the 20 (top) and 100 (bottom) game training sets. Bands represent the standard deviation. Note that on level 0, training curves for TDQN, GATA$^\text{C}$, and GATA$^\text{D}$ were early stopped for achieving ≥ 0.95 normalized accumulated reward on the validation set for 5 episodes in a row.

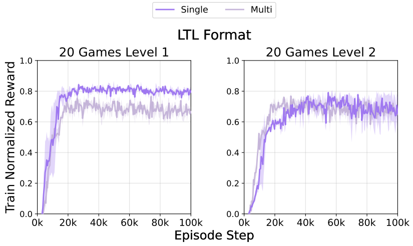
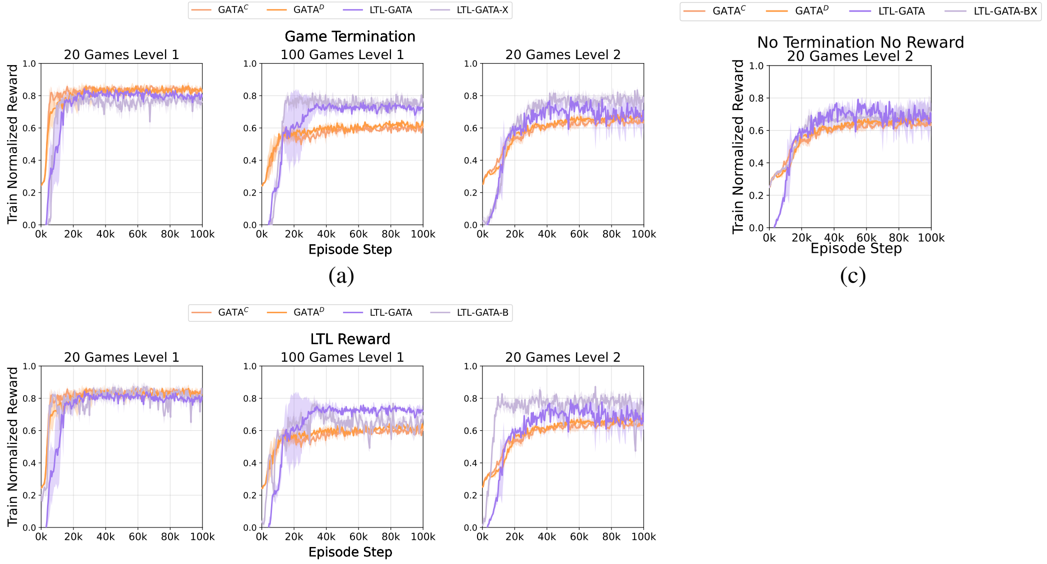


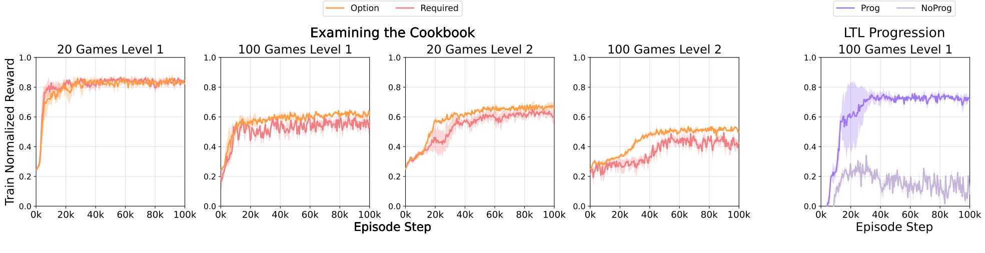
Figure 10: Training curves (of normalized accumulated reward) for the study on LTL predicate format with single-token (*Single*) and multi-token (*Multi*) predicates. Bands represent the standard deviation.



(a) (c)

(b)

Figure 11: Training curves (of normalized accumulated reward) for the ablation studies on LTL-based episode termination and new reward function $R_{\text{LTL}}(s, a, \varphi)$. (a) LTL-GATA with new reward function $R_{\text{LTL}}(s, a, \varphi)$ and without LTL-based episode termination (*LTL-GATA-X*). (b) LTL-GATA with base game reward function $R(s, a)$ and with LTL-based episode termination (*LTL-GATA-B*). (c) LTL-GATA with base game reward function $R(s, a)$ and without LTL-based episode termination (*LTL-GATA-BX*). Bands represent the standard deviation.



(a) Letting GATA Consume the Cookbook

(b) Progression Ablation

Figure 12: Training normalize reward curves for (a) comparison of GATA^{D} when given the *Option* to examine the cookbook vs. when it is *Required* to examine the cookbook and (b) comparison of LTL-GATA with (*Prog*) and without (*NoProg*) using LTL progression. Bands represent the standard deviation.

H.6 Automated Translation: Natural Language Instructions to LTL Details

For the GPT-3 experiments on automated LTL translation in subsection 5.5, we simply extracted the observations used by our simple translator and saved the observation-translation pairs. Details of that translator were described in in Appendix E and examples of these pairs can be found in Table 3 and Table 4. These observations are the sort of natural language we wish to translate into LTL, and we used six of the observation-translation pairs as the examples in our prompt to GPT-3.

The full prompt to GPT-3 is shown below (with colors added for readability). The six examples consist of a natural language observation (in turquoise) and a corresponding LTL formula (in red) — these remain fixed for all prompts. The seventh line begins with the natural language observation to be translated (in blue).

1. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : cilantro directions : dice the cilantro prepare meal LTL: (‘and’, (‘eventually’, ‘cilantro_in_player’), (‘and’, (‘eventually’, ‘cilantro_is_diced’), (‘eventually’, ‘meal_in_player’)))

2. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : pork chop directions : chop the pork chop fry the pork chop prepare meal LTL: (‘and’, (‘eventually’, ‘pork_chop_in_player’), (‘and’, (‘eventually’, ‘pork_chop_is_chopped’), (‘and’, (‘eventually’, ‘pork_chop_is_fried’), (‘eventually’, ‘meal_in_player’))))

3. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : black pepper directions : prepare meal LTL: (‘and’, (‘eventually’, ‘black_pepper_in_player’), (‘eventually’, ‘meal_in_player’))

4. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : purple potato red onion salt directions : dice the purple potato roast the purple potato dice the red onion fry the red onion prepare meal LTL: (‘and’, (‘eventually’, ‘purple_potato_in_player’), (‘and’, (‘eventually’, ‘red_onion_in_player’), (‘and’, (‘eventually’, ‘salt_in_player’), (‘and’, (‘eventually’, ‘purple_potato_is_diced’), (‘and’, (‘eventually’, ‘purple_potato_is_roasted’), (‘and’, (‘eventually’, ‘red_onion_is_diced’), (‘and’, (‘eventually’, ‘red_onion_is_fried’), (‘eventually’, ‘meal_in_player’))))))))

5. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : black pepper parsley salt directions : dice the parsley prepare meal LTL: (‘and’, (‘eventually’, ‘black_pepper_in_player’), (‘and’, (‘eventually’, ‘parsley_in_player’), (‘and’, (‘eventually’, ‘salt_in_player’), (‘and’, (‘eventually’, ‘parsley_is_diced’), (‘eventually’, ‘meal_in_player’)))))

6. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : purple potato white onion yellow bell pepper directions : roast the purple potato roast the white onion dice the yellow bell pepper prepare meal LTL: (‘and’, (‘eventually’, ‘purple_potato_in_player’), (‘and’, (‘eventually’, ‘white_onion_in_player’), (‘and’, (‘eventually’, ‘yellow_bell_pepper_in_player’), (‘and’, (‘eventually’, ‘purple_potato_is_roasted’), (‘and’,

(‘eventually’, ‘white_onion_is_roasted’), (‘and’, (‘eventually’, ‘yellow_bell_pepper_is_diced’), (‘eventually’, ‘meal_in_player’)))))))

7. NL: you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : banana red hot pepper yellow potato directions : chop the banana fry the banana chop the red hot pepper fry the red hot pepper slice the yellow potato fry the yellow potato prepare meal LTL:

In Table 7, we show how *Da Vinci* and *Ada* complete the prompt to translate this example of a natural language TextWorld observation. While *Da Vinci* produces the correct LTL translation, *Ada* struggles and hallucinates incorrect predicates such as `‘zucchini_fry_player’` and `‘banana_is_frozen’`.

Table 7: GPT-3 Natural-Language-to-LTL Translation Example

Observation
you open the copy of “cooking : a modern approach (3rd ed .)” and start reading : recipe # 1 ----- gather all following ingredients and follow the directions to prepare this tasty meal . ingredients : banana red hot pepper yellow potato directions : chop the banana fry the banana chop the red hot pepper fry the red hot pepper slice the yellow potato fry the yellow potato prepare meal
***Da Vinci* LTL Translation (*Absolutely Correct*)**
(‘and’, (‘eventually’, ‘banana_in_player’), (‘and’, (‘eventually’, ‘red_hot_pepper_in_player’), (‘and’, (‘eventually’, ‘yellow_potato_in_player’), (‘and’, (‘eventually’, ‘banana_is_chopped’), (‘and’, (‘eventually’, ‘banana_is_fried’), (‘and’, (‘eventually’, ‘red_hot_pepper_is_chopped’), (‘and’, (‘eventually’, ‘red_hot_pepper_is_fried’), (‘and’, (‘eventually’, ‘yellow_potato_is_sliced’), (‘and’, (‘eventually’, ‘yellow_potato_is_fried’), (‘eventually’, ‘meal_in_player’))))))))))
Ada* LTL Translation *(Incorrect)
(‘and’, (‘eventually’, ‘banana_in_player’), (‘and’, (‘eventually’, ‘red_hot_pepper_in_player’), (‘and’, (‘eventually’, ‘yellow_potato_in_player’), (‘and’, (‘eventually’, ‘zucchini_fry_player’), (‘and’, (‘eventually’, ‘banana_is_frozen’), (‘eventually’, ‘meal_in_player’))))

I Broader Impact

As Adhikari et al. (2020) suggested, text-based games can be a proxy for studying human-machine interaction through language. Human-machine interaction and relevant systems have many potential ethical, social, and safety concerns. Providing inaccurate policies or information or partially completing tasks in critical systems can have devastating consequences. For example, in health care, improper treatment can be fatal, or in travel planning, poor interactions can lose a client money.

Adhikari et al. (2020, section 7) identified several research objectives relating to language-based agents: improve the ability to make better decisions, allow for constraining decisions for safety purposes, and improve interpretability. We highlight how RL agents equipped with LTL instructions can improve in these areas. For constraining decisions, it may be desirable to do so in way that depends on the history, which LTL gives a way to keep track of. With respect to interpretability, we propose that monitoring the progression of instructions provides a mechanism for understanding where and when an agent might be making incorrect decisions, and provides the opportunity to revise instructions or attempt to fix the problem by other means.

However, instruction following, especially overly literal instruction following, may not always be beneficial and can even be harmful. Ammanabrolu et al. (2022) describe a good example where an agent in the Zork1 game breaks into a home and steals the items it needs. In that specific case, breaking into the home has no adverse effect on the agent's reward, and so it has no incentive not to perform this act. Violation of social norms like this are not modelled in our work, and can have negative impacts, even in less extreme cases. Furthermore, there are potential dangers of incorrect, immoral, or even misinterpreted instructions that lead to dangerous outcomes. Although we do not directly address these concerns in this work, they pose interesting directions for future work.

Additional References

Ammanabrolu, P., Jiang, L., Sap, M., Hajishirzi, H., and Choi, Y. Aligning to social norms and values in interactive narratives. *CoRR*, abs/2205.01975, 2022. doi: 10.48550/arXiv.2205.01975. URL `https://doi.org/10.48550/arXiv.2205.01975`.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond. 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=rkgz2aEKDr`.