Interactive Question Answering

A Sequence Modelling Approach to Question-Answering in Text-Based Games

Gregory Furman
FRMGRE001@myuct.ac.za
UCT Computer Science Honours

Abstract

Interactive Question Answering (IQA) has been proposed to solve current systems failing to develop comprehension abilities. To this end, the Question Answering using Interactive Text (QAit) task was created to produce and benchmark interactive agents seeking information and answering questions in a procedural knowledge setting. While prior work has exclusively focused on IQA as a reinforcement learning problem, such methods suffer from low sample efficiency and poor accuracy on zero-shot evaluation. In this paper, we propose framing an IQA trajectory as a sequence modelling problem. Using the novel *Decision Transformer* architecture, we investigate the applicability of a Transformer architecture in modelling IQA problems at scale. By utilising a causally masked GPT-2 Transformer for action generation and a BERT model for answer prediction, we show the Decision Transformer achieves performance greater than or equal to current state-of-the-art RL baselines on the QAit task in a more sample efficient manner.

1 INTRODUCTION

Recent work has shown that question-answering (QA) and machine reading comprehension (MRC) systems fail to develop the necessary *comprehension* abilities required to fulfil a question-answering task [21, 26, 27]. Traditional methods for QA and MRC are primarily concerned with the retrieval of *declarative knowledge*, that is, explicitly stated or static descriptions of entities within a knowledge base (KB) [26]. In addition, these models tend to cultivate basic pattern matching skills, further differentiating their abilities from those of humans [26]. Conversely, *procedural knowledge* is the sequence of actions required to perform a task [9]. To this end, researchers propose interactive question-answering (IQA) as a means of teaching MRC systems to gather information necessary for question answering [26].

IQA requires an agent to interact with some dynamic environment in order to gather the knowledge necessary to answer a question [10]. Given the interactiveness, such a task is primarily suited towards a reinforcement learning (RL) based solution. Yuan et al. [26] proposed Question Answering using Interactive Text (QAit) as a means of testing the knowledge gathering capabilities of an agent required to answer a question about the environment with which it inhabits. Here an agent interacts with a partially observable text-based environment, created using Microsoft TextWorld [7], in order to gather information and answer questions about the attributes, location, and existence of objects. The QAit task aims to produce interactive agents seeking information and answering questions in a procedural knowledge setting.

The results achieved by Yuan et al. suffered from low-sample efficiency and relatively poor performance on existence and attribute question types. These shortcomings necessitate investigation into current architecture and methodology used by QAit and motivates the exploration of alternatives. To this end, we hypothesise that a transformer architecture that treats a reinforcement learning trajectory as a sequence of states, actions, and rewards would yield significant improvements over the current QAit baseline.

Transformers [23] have shown success in modelling a diverse range of high-dimensional problems at scale [4, 8, 20]. The ability to employ transfer-learning and fine-tuning also means that existing models such as Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-Trained (GPT) [19] models can be utilised and trained in a stable manner. Hence, the aforementioned benefits coupled with a demonstrated ability for transformers to model long sequences by utilising the self-attention mechanism makes this architecture ideal for tasks of IQA. Recent works by Chen et al. [5] & Janner et al. [13] have shown the applicability of such models to sequential decision-making problems offering an alternative solution to RL based problems. The authors propose framing a given RL trajectory as a sequence of states, actions, and rewards modelled autoregressively by a Transformer. This sequence modelling transformer for RL based problems will be referred to as a *Decision Transformer* (DT) [5].

In this paper, we propose replacing the original QA module with a fine-tuned BERT model, aiming to leverage pre-trained word embeddings and language understanding to provide more accurate answers to questions. We will replace the role of the RL agent with a Decision Transformer that utilises the GPT-2 [19] architecture and will closely follow the methodology outlined by Chen et al. [5]. In this paper, we investigate two hypotheses:

(1) By framing the QAit task as a sequence modelling problem, a Decision Transformer will outperform previous benchmarks set out by Yuan et al. with respect to the gathering of procedural information required to answer a question, referred to as *sufficient information*. The Decision Transformer will benefit from increased sample efficiency and speed of training over previous RL methods tested. Moreover, despite using sub-optimal data generated via random rollouts, previous benchmarks will be matched and outperformed. As QAit is considered sparsely-rewarded [26], particularly attribute type questions, we posit that the Decision Transformer's reward-agnostic abilities will lead to improved performance [5].

(2) Replacing the current QA module with a BERT model fine-tuned to the QAit task will improve question answering capabilities over prior methods. This paper posits that leveraging the BERT model's bidirectionality and pre-trained language embeddings can improve sample efficiency and QA accuracy.

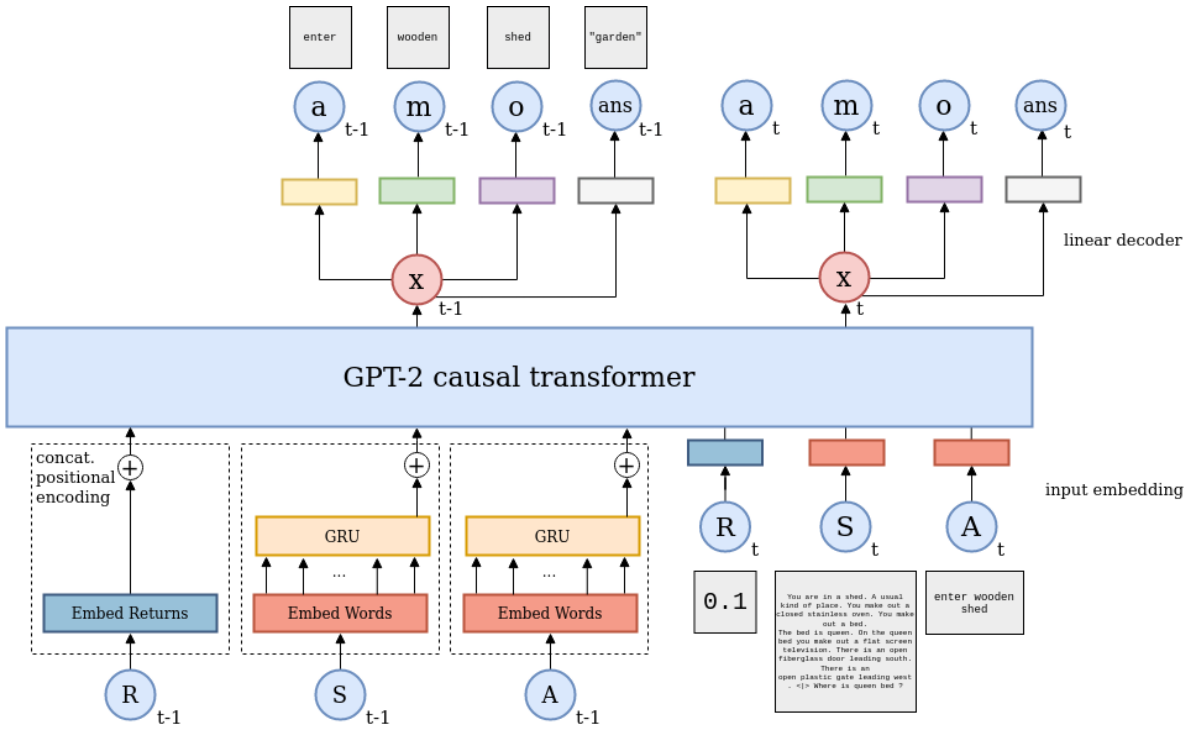

Figure 1: Decision Transformer architecture. States, Actions, and Returns are each fed as input into modality specific linear layers with a positional episodic timestep encoding being added. Tokens are fed into a GPT-2 architecture allowing for the autoregressive prediction of actions, called *commands*, using a causal self-attention mask. Action predictions are then fed into three linear-decoders each representing the action, modifier, and object of a command. States and actions are projected to the embedding dimension and encoded by a GRU. An embedding is also learnt for returns-to-go. All three tokens are then concatenated with a positional embedding for the episodic timestep t. The output of the GPT Decision Transformer is fed into linear decoders for the prediction of a command's action, modifier, and object components as well as a decoder for predicting the answer to the question. Diagram & description adapted from Chen et al. [5].

2 BACKGROUND

2.1 Text-based Environments

Dynamic environments can be created using a text-based world generating framework that RL agents can interact with to achieve some goal - commonly rooted in information extraction or question answering. The underlying environment with which the agent inhabits can be some abstraction of a real-world KB wherein different in-game objects or locations can be decoded as representing some real-world entity-relation pairs. Agents are expected to learn optimal policies within this textual space. These policies represent some abstraction of "skills" required to perform a sequence of actions to maximise a reward. Such skills include certain forms of reasoning, comprehension, memory, as well as contextual-awareness of their environment [18]. One of the most popular framework for text-based games is *TextWorld* [7, 18]. TextWorld is an open-source simulator developed by *Microsoft* that aims to train RL agents to acquire skills such as decision making and language comprehension[1]. With the popularity of text-based games, many of the barriers to entry for approaching RL problems in NLP have been lifted - which some posit will have a significant impact on language learning in dialogue-like environments [14, 18].

2.2 QAit Task

Using TextWorld, Yuan et al. developed Question Answering with Interactive Text (QAit) [26]. The QAit task measures an agent's procedural and declarative knowledge-gathering abilities required to answer a question about an interactive text-based environment. Developed atop Microsoft TextWorld [7], QAit expects an agent to answer questions requiring an understanding of locality, existence, and attributes of objects within an environment. Such environments are procedurally generated via sampling from a distribution of world settings.

2.2.1 TextWorld Environment An agent interacts with a TextWorld environment using textual commands that consist of an action, modifier, and object triplet e.g "open conventional oven". An environment consists of a fixed number of rooms, each containing randomly assigned objects and location names. Navigating rooms and interacting with objects is done purely using text-based commands. Upon an agent giving a command, TextWorld will respond with a state-string

[1] https://www.microsoft.com/en-us/research/project/textworld/ (Accessed June 3 2021)

showing the consequence of that action. State strings contain information about the room an agent is in and the objects that are currently present. Environments are of two map types: fixed map and random map. A fixed map contains six rooms, whereas random maps have their number of rooms sampled from a uniform distribution $U(2, 12)$.

2.2.2 Question types An agent is required to answer one of three question types: location, attribute, and existence. All questions relate to object characteristics.

- **Location questions** assess an agent's ability to understand the location of objects within the environment. Here, an agent navigates the world trying to find a desired object and answer which container it is in. For example, "Where is copper key?" could be answered with "garden" or "toolbox".
- **Existence questions** relate to the existence of objects within the environment. Here, an agent moves about and interacts with entities in the world to gather knowledge and determine whether an object exists. Answers are either yes ("1") or no ("0"), with questions being phrased as "is there any X in the world?", where X is an entity in the vocabulary.
- **Attribute questions** are the most difficult to answer as they require high interactivity and movement throughout the world. Similarly to existence, attribute type answers are either yes or no. These question types necessitate an agent to comprehend both a question and the environment. Thus, these are answered with the lowest accuracy of all three types. Moreover, entities in question often have arbitrary names and attributes, making memorisation impossible. For example, "Is stove hot?" requires an agent to find and interact with "stove" to answer the question correctly.

2.2.3 Games Yuan et al.'s methodology for evaluating agent performance sees average QA training accuracy calculated over five training sets. These are fixed and random map types having Number of Games settings of 1, 2, 10, 100, 500, and unlimited, respectively. Unlimited games see worlds generated on the fly during training. The number of games indicates the number of unique environments that an agent will interact with during training.

2.2.4 Rewards & metrics In order to aid agent learning, Yuan et al. proposed two rewards to be used in QAit environments to help the learning process:

- **Sufficient Information Reward** measures how much information an agent has gathered that is required to answer the question. This value is awarded at the end of an episode and not given at each timestep.
- **Exploration Reward** is awarded to an agent whenever entering a previously unseen state in order to promote exploration of the environment. Interchangeably referred to as exploration bonus.

2.3 Evaluation

An agent's accuracy will be measured using zero-shot evaluation where five hundred never-before-seen games are held out during training, each containing a single question. Yuan et al. proposed this as a means of assessing a model's generalisation abilities in a manner akin to the test set of supervised learning problems.

The QAit task measures an agent's performance by sufficient information and QA accuracy. The former quantifies the amount of procedural knowledge required to answer a question that an agent has gained. The latter indicates the percentage of correct answers to questions about the environment. The heuristics for sufficient information calculation can be seen in the original QAit paper as well as Table 8.

2.4 Sequence modelling

2.4.1 RNN Recurrent neural networks (RNNs), such as long short-term memory (LSTM) [12] or Gated Recurrent Unit (GRU) models [6], are commonly associated with sequence modelling problems and, until recently, such models were considered state-of-the-art [22]. However, such models have several major shortcomings. Due to the vanishing gradient problem [17], as a sentence increases in length, the likelihood of maintaining context decreases exponentially. Although LSTMs and GRUs can retain some memory to mitigate this, through the usage of gated units, long-term dependencies are still forgotten [22]. Moreover, as each hidden state h_t is a function of the previous hidden state h_{t-1} and the input position t, increasing performance through parallelisation is functionally impossible [23].

2.4.2 Transformers Vaswani et. al. [23] proposed Transformers as an efficient architecture to model sequential data. These models outperformed the training costs and model accuracy of convolutional and RNN approaches to common language modelling and translation problems [23]. The authors show that an attention mechanism can entirely replace recurrence and convolutions resulting in an architecture consisting entirely of stacked self-attention layers with residual connections. Transformers can model global dependencies and offer significant speedup capabilities via parallelism by eschewing recurrence and relying solely on an attention mechanism.

2.4.3 Attention mechanism A Transformer's self-attention mechanism, described by Chen et. al. [5], sees each self-attention layer be fed n embeddings $\{x_i\}_{i=1}^{n}$ with each embedding being directly related to some input tokens. The i^{th} token is mapped via a linear transformation to a key k_i, query q_i, and value v_i. As output, each layer returns n embeddings $\{z_i\}_{i=1}^{n}$, where input dimensionality has been preserved. Output i of the self-attention layer is calculated by weighting values v_j by the normalised dot product between query q_i and key k_j:

$$z_i = \sum_{j=1}^{n} softmax(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \cdot v_j$$

2.5 Decision Transformer

Proposed by Chen et al. [5], the Decision Transformer architecture, see Figure 1, aims to autoregressively model a trajectory of actions, states, and rewards using a Transformer architecture, specifically GPT-2 [19]. The authors show the DT to perform as well as, if not better than, current state-of-the-art model-free RL algorithms on Atari [1], OpenAI Gym [3], and Key-to-Door [16] tasks [5]. They also show the DT's ability to outperform RL baselines in both sparsely rewarded tasks and problems where long-term credit assignment is required. See Section 4.3 for further details.

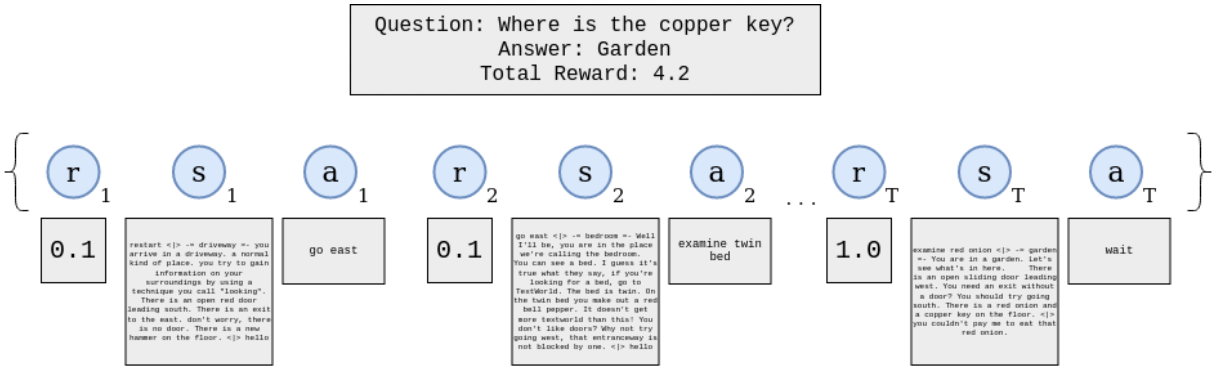

Figure 2: Example trajectory in the form $(r_1, s_1, a_1, r_2, s_2, a_2 ... r_T, s_T, a_T)$ with additional data such as the total reward gained, the question to be asked, and the final correct answer to the question.

2.5.1 Trajectory The Decision Transformer architecture, see Figure 1, aims to autoregressively model a trajectory of actions, states, and rewards. After every episode play-through, the entire trajectory is processed and stored in the form $(r_1, s_1, a_1, r_2, s_2, a_2 ... r_T, s_T, a_T)$ where r refers to action's reward, s is a state, and a is an action. This can be seen in Figure 2. The trajectory representation enables simple autoregressive training where the autoregressive GPT model can be conditioned on reward and starting state to generate the desired action sequence. As action prediction is based on gaining some future reward, rather than how much reward has already been gained, Chen et al. [5] propose conditioning actions upon the total reward that can still be gathered from interacting with the environment. This is referred to as the returns-to-go (RTG) $R_t = \sum_{t'=t}^{T} r_{t'}$ where T is trajectory length and r_t is the reward at time step t. Thus the initial return-to-go R_1 represents the total reward to be gained from a given episode.

2.5.2 Reward If an agent gains some reward while interacting with the world, this will be deducted from its returns to go from that timestep onwards. Thus, when feeding input to the DT, this is in the form $(R_1, s_1, a_1, R_2, s_2, a_2 ... R_T, s_T, a_T)$. To illustrate this concept, Figure 2 has a total reward of 4.2. During training, we know the first return-to-go R_1 will be set to 4.2, as this is the total reward to be gained from the entire trajectory. Next, R_2 will be 4.1 as a reward of 0.1 was given at $t = 2$. Finally, at step T, R_T will be 0.0, as there is no longer any reward to be gained.

3 RELATED WORK

3.1 Trajectory Transformer

Concurrently to work by Chen et. al. [5], Janner et. al. proposed [13] the Trajectory Transformer (TT). At its core, the authors propose treating a trajectory as an unstructured sequence modelled by a Transformer architecture. As such, the architecture of the Trajectory Transformer closely resembles that of the Decision Transformer, albeit with minor differences. The Trajectory Transformer is trained in a manner akin to a natural language processing task [13], which seeks to model the joint probability distribution over a sequence of states, actions, and rewards. During training, the standard teacher-forcing [24] procedure is used with the dynamic-programming Beam-Search [11] algorithm for decoding and autoregressive prediction. This method is in contrast to the DT, where the authors combine the tools of sequence modelling with hindsight return information in order to achieve policy improvement **without the need for dynamic programming** [5]. Both methods utilise the GPT architecture [19] with the Trajectory Transformer having six self-attention heads and four layers, and the Decision Transformer having eight self-attention heads and two layers. On standard RL baselines, the DT and TT yielded relatively similar results [5, 13] - but due to the recency of both frameworks, no head-to-head comparison has been conducted.

4 METHODS

4.1 QAit Task

For this paper, we will be training the Decision Transformer on trajectories generated from the 500 games fixed and random map type settings. A key research outcome of this paper is to illustrate the Decision Transformer's sample efficiency and high accuracy on QAit's zero-shot evaluation test set. Yuan et al. [26] showed that those highest scoring agents, with respect to QA accuracy, were predominantly trained from the *unlimited* games setting. We, therefore, opt to generate offline data from the 500 games setting over *unlimited* games as a means of purposefully providing the DT with suboptimal data in the hopes that it will outperform previous baselines - thereby illustrating our hypothesis.

4.2 Random rollouts

The key success of the Decision Transformer is the ability to learn effective behaviour from fixed and limited experiences [5]. This ability further motivates its applicability to the QAit task, where training is time inefficient and resource-intensive. However, converting this RL problem into a supervised learning problem requires training data ahead of time. Therefore, we generate datasets of offline RL data using *random rollouts* for each map and question-type, holding the number of games constant at 500. Statistics about the training data are available in Table 1. We generated Random rollouts using a random agent who samples actions from all *admissible commands*

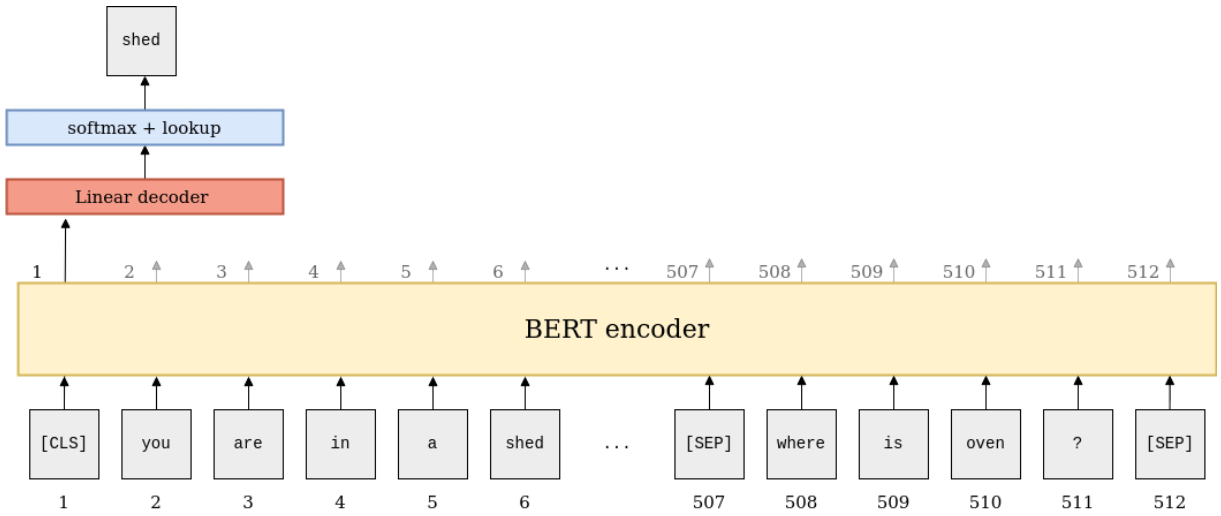

Figure 3: The QA module consists of a BERT encoder that takes as input a concatenation of multiple state descriptions. The encoder output is feeds into a linear decoder that returns a vector where each element corresponds to a token in the QAit vocab. This vector is then softmaxed and used to determine the most likely word in the vocab that answers the question.

for a particular gamestep. The sequence of states observed and actions performed are recorded in an offline dataset, along with the rewards for each action. Moreover, questions and answers are also added to the offline trajectory.

Given that the vocabulary consists of 1654 unique tokens, there are roughly 1654^3 possible commands at any given timestep - most of which are nonsensical and unable to be registered as valid by TextWorld. The sparsity of the action space means that generating commands by randomly selecting an action, modifier, object triple from the vocabulary would lead to largely nonsensical actions, thereby motivating the selection of random but valid commands from the admissible actions at step t. By using admissible commands in data generation, the Decision Transformer will have the ability to learn admissible commands from suboptimal data.

Map Type	Question Type	Mean Reward	Max Reward	Commands	N
Fixed	Location	0.526	4.1	7.631	44k
	Existence	0.554	3.8	8.489	39k
	Attribute	0.498	3.729	7.606	36k
Random	Location	0.565	4.1	7.693	41k
	Existence	0.606	3.944	8.91	42k
	Attribute	0.542	4.033	7.731	82k

Table 1: Average and maximum total reward gained from training set. Additionally, the average number of unique admissible commands per trajectory is shown to illustrate the action space from which the DT learns.

4.3 Decision Transformer

4.3.1 Architecture The Decision Transformer architecture can be fed the last K timesteps as input allowing for K returns-to-go, actions, and state tokens, respectively (totalling $3K$ tokens per timestep t). The parameter K is a context window consisting of the number of previous episodes the transformer can draw upon to inform its decision-making. We set the maximum length of a QAit episode to 50 episodic timesteps as well as $K = 50$. This allows for a maximum of 50 timesteps (or maximum length of a trajectory) to be fed as input to the Decision Transformer at a time.

Token embeddings for states and actions are obtained using a single embedding layer wherein the raw token inputs are projected to an embedding dimension. Given that states and actions can be of variable length, these embeddings are fed to a GRU encoder with the final hidden state h_n representing the entire encoded state or action. Embeddings for returns-to-go R_t are also learnt and projected to the embedding dimension. Finally, an embedding for each episodic timestep t is concatenated to each embedded token. The embedded and positionally encoded action, state, and return-to-go inputs are fed into the GPT model. This input architecture can be seen in Figure 1.

For any given timestep t, the Decision Transformer output x_t is fed into four linear decoders. Three correspond to a command's action, modifier, and object components with the fourth additional linear decoder serving to predict the answer to the question at each timestep. While not acting as the primary QA mechanism in the architecture, the answer decoder allows the Decision Transformer to learn some primitive level of question-answering in conjunction with command prediction to integrate question-answering with command generation. See Figure 1 and Algorithm 1 for more details.

We also limited the size of input states to a length of 180 tokens, as seen in Table 2. Similar to the methodology found in Section 4.4.1, we concatenated the question to the end of a state string and separated both by a "<|>" token in the form `[STATE] <|> [QUESTION]`.

Table 2: Figure showing hyperparameters used when training the Decision Transformer and the BERT QA module. For the DT, Context length *K* refers to the amount of previous timesteps with which the Transformer can draw upon to navigate the environment. Context State context window refers to the number of tokens from the state to be used for prediction. Adam [15] is used as an optimiser in conjunction with the specified learning rate and weight decay.

	Decision Transformer	BERT QA module
Hyperparameters	**Value**	
Number of layers	2	
Number of attention heads	8	
Embedding dimension	256	64
Batch size	128	12
State context window	180 tokens	512 tokens
Context length (K)	50	-
Max Epochs	2000	30
Dropout	0.5	0.1
Learning rate	1×10^{-4}	1×10^{-5}
Adam betas	(0.9, 0.95)	
Grad norm clip	0.25	
Weight decay	0.1	
Learning rate decay	Linear warmup and cosine decay	

4.3.2 Training As actions and answers are discrete, the categorical cross-entropy loss of the action, modifier, object, and answer prediction is used for training. Chen et al.'s [5] findings indicated predicting state and returns-to-go at each timestep did not have a positive effect on performance, motivating us to follow suit and only predict action. During training time, R_1 is set to the total possible reward to-be-gained from a particular trajectory. This total reward value is available as training data consists of completed sequences of offline trajectories.

When evaluating, however, the true R_1 cannot be known a priori. To reconcile this, R_1 is either set as a fixed value or sampled from an exponential distribution. For each question and map type configuration, a set of unique evaluation games were generated wherein an agent must interact with an environment to answer a question. During training, the Decision Transformer is tested in the appropriate set of hold-out games every 250 iterations to monitor sufficient information scores and overfitting.

4.3.3 Action Generation Instead of greedily decoding [25], we softmax the output of each action, modifier, and object prediction head, creating a probability distribution that is randomly sampled from for action generation. The additional randomness, from randomly sampling over a probability distribution, minimises the risk of the Decision Transformer entering an action loop - in which the same command is generated continually. It also serves to better mirror natural language, where phraseology and word-choice is non-deterministic. However, the output of the answer prediction head is deterministic, wherein we argmax its output to get the answer index from the vocab.

4.4 QA module

4.4.1 Architecture The QA module, see Figure 3, consists of a classification layer sitting atop a BERT encoder. A benefit of using a BERT model is that multiple state strings can be used as context in order to answer a question. First, states are joined together into a single long sequence of tokens with the question appended at the end. This concatenated state-and-question string is tokenised by a pretrained *bert-base-uncased* tokeniser. This pads or truncates the input returning a 512 token long representation then fed to the BERT encoder. This tokeniser also separates the prompt and question using the `[SEP]` tag as well as adds `[CLS]` and `[SEP]` to the beginning and end of the string, respectively. Finally, we pass BERT's pooled output to a linear layer used to predict a word from the vocab that answers the question in a manner akin to a classification task. The classifier has two output nodes for attribute and existence questions, one for "yes" or "no", respectively. For location type questions, the classifier has a node for each token in vocabulary, making it identical to the answer prediction mechanism of the Decision Transformer.

4.4.2 Training Training and validation sets are created that consist entirely of valid trajectories. We use the validation set to save the model with the highest out-of-sample QA accuracy. We use categorical cross-entropy to calculate loss between predicted and actual answers. In order to gather the state-strings necessary to train the QA module, we feed the random rollouts, used for training the DT, back into the DT and allow for it to predict when to stop. We then take the last 512 tokens of the trajectory's states, from when the DT predicted to halt, and use that as input to train the QA model.

To prevent overfitting, 20% of all offline trajectories used in the training set are held-out for validation each iteration. The model with the highest validation accuracy after all 30 epochs of training is thus saved.

4.5 Validation & Tuning

We created a hold-out validation set of 50 games to evaluate the question-answering performance of the DT and BERT model. As previously discussed, initial returns-to-go R_1 cannot be known a priori, motivating us to test different R_1 when evaluating the DT.

Table 3: QA accuracy and sufficient information score (in brackets) of each model following zero-shot evaluation on the test set in the 500 games setting. A bolded value indicates a score to be the highest of that question and map type configuration for the 500 games setting. An asterisk (*) indicates a model has the highest accuracy of all game configurations in the QAit task (see Section 2.2.3). See training accuracies in Tables 6 and 7.

Model	Location		Existence		Attribute	
	Fixed	Random	Fixed	Random	Fixed	Random
DQN	0.224(0.244)	0.204(0.216)	**0.674(0.279)**	0.678(0.214)	**0.534(0.014)***	0.530(0.017)
DDQN	0.218(0.228)	0.222(0.246)	0.626(0.213)	0.656(0.188)	0.508(0.026)	0.486(0.023)
Rainbow	0.190(0.196)	0.172(0.178)	0.656(0.207)	0.678(0.191)	0.496(0.029)	0.494(0.017)
DT	0.168(0.232)	0.104(0.264)	0.668(0.254)	**0.722(0.277)***	0.504(0.057)	0.526(0.058)
DT-BERT	**0.232(0.232)**	**0.270(0.264)***	0.626(0.258)	0.654(0.277)	0.524(0.058)	**0.538(0.060)**

Specifically, this allows us to investigate performance in sparsely and richly rewarded environments and the extent to which this affects the DT's predictive, navigational, and knowledge-gathering capabilities. We treat this initial value as a hyperparameter and test a wide range of values to determine the optimal starting value for each question and map type in a manner akin to a grid-search. Similar to methodology used by Yuan et al. [26] for selecting the best model during training, we opt to determine the R_1 value based on the maximum combination of sufficient information score and question answering accuracy. The discussion for this is available in Section A of the Appendix.

4.6 Testing

We determine a model's final performance using zero-shot evaluation on a test set provided by QAit. This set consists of 500 unseen games that aim to test an agent's question-answering, knowledge gathering, and generalisability skills and was used to benchmark all previous methods abilities in the QAit task. First, we evaluate the Decision Transformer's action generation and answer prediction abilities on the test-set to assess its performance against previous results. Next, we will replace the Decision Transformer's answer prediction module with the fine-tuned BERT QA model (DT-BERT) and evaluate it on the same set.

5 RESULTS

The following sections will examine the Decision Transformer and BERT question-answering model. We show the QA results for the BERT model in Figure 5 and the DT's answer prediction head in Figure 6 for every hyperparameter R_1 tested. The final results are available in Table 6 for fixed map types and 7 for random. The final QA accuracies are also shown in Table 3 and visualised in Figure 4.

5.1 Location Questions

On the test set, we found the DT's answer prediction head to have a QA accuracy of 0.104 on random map games and 0.168 on a fixed map. Thus, while outperforming the random baseline, the DT's answer prediction capabilities did not surpass previous RL methods for the 500 games setting, both for fixed and random map types. However, we found that the information gathering capacity of the DT surpassed all prior approaches to location type questions, having scored 0.264 on random map and 0.232 on fixed map settings.

The DT's question-answering capacity was seemingly decoupled from its knowledge gathering abilities. Findings by Yuan et al. [26] showed the ability of an agent to gather information was closely associated with its ability to answer location questions, where sufficient information scores were closely related to question-answering accuracy. This disconnect between the high sufficient information scores of DT and its relatively poor question-answering abilities was likely a result of the answer prediction head underfitting the training data.

The BERT QA model achieved a QA score of 0.27 in with a sufficient information score of 0.264 in the random map setting. In a fixed map, the model scored a QA accuracy of 0.232 with a sufficient information score of 0.232. Thus, the BERT model outperformed the QA accuracy of all models trained in the 500 games setting in fixed and random maps. The QA accuracy mirroring that of the sufficient information score indicates perfect performance on questions of the locality of objects. For fixed map questions, the BERT model's accuracy is seemingly limited by whether or not the agent manages to navigate into the correct state, explaining the equal scores. However, the BERT model learnt to use additional context to achieve a higher QA accuracy than its sufficient information score for the random map settings. This decoupling of the BERT QA model and the DT means that a question can still be correctly answered even if the DT stops in an incorrect state. This high accuracy mirrors the results of the held-out validation set used when training the BERT QA model, seen in Table 4.

5.2 Existence Questions

The DT scored a QA accuracy of 0.722 and a sufficient information score of 0.277, outperforming all previous sufficient information and QA accuracy baselines in the 500 games random setting (see fixed map baselines in Table 6 for and random map in Table 7 for the 500 games settings). However, while outperforming the DDQN and Rainbow, the DT failed to outperform the DQN in the fixed map setting, achieving an accuracy of 0.668 and a sufficient information score of 0.254.

For the BERT model, zero-shot evaluation on the test set resulted in a QA accuracy of 0.654 for random map and 0.626 for fixed map. Thus, despite having shown promise on the hold-out set during training, the BERT QA model could not outperform the DT's answer prediction head and the previous QA baselines in both the fixed map and random map settings. While achieving well above the random

Table 4: QA accuracy of the BERT model on the 20% of held-out trajectories during training.

Question	Map Type	Max Validation Accuracy
Attribute	Fixed	0.780
	Random	0.616
Existence	Fixed	0.778
	Random	0.779
Location	Fixed	0.987
	Random	0.988

baseline, the reasons for this underperformance are likely a result of overfitting the training set.

5.3 Attribute Questions

The Decision Transformer scored higher sufficient information on the test set than all previous RL baselines, achieving 0.058 in random map and 0.057 in fixed map. Additionally, the QA accuracy of the answer prediction head surpassed the DDQN and Rainbow in the random map 500 games setting, with a QA accuracy of 0.526 - but failed to outperform the DQN. For fixed map, similar results were observed wherein the DT outperformed the DDQN and Rainbow, scoring 0.504 but failed to beat the DQN. Despite the superior knowledge-gathering ability of the DT, its question-answering capacity was unable to outperform the prior QA module's maximum accuracy of 0.530 and 0.534 for 500-games random and fixed map settings, respectively.

During zero-shot evaluation on the test set, the BERT QA model scored similarly to the baselines set by Yuan et al., beating the random map baselines but failed to outperform the fixed map results in the 500 games setting. While BERT outperformed Rainbow and the DDQN in the fixed map setting, scoring 0.524, the QA abilities of the DQN were unable to be surpassed. On the other hand, the BERT model outperformed all baselines in the 500 games random map setting, achieving the highest sufficient information score of the attribute test set and a QA accuracy of 0.538. In both settings, we attribute the success of DT-BERT to the BERT model's leveraging of pre-trained language embeddings.

6 DISCUSSION

6.1 Reward and Performance

6.1.1 Location The optimal initial return-to-go for location type questions was determined to be 2.0 for both fixed and random map settings, meaning exploration is not as highly encouraged as with the existence and attribute types. In location type questions, the entity definitively exists somewhere within the environment. This means that the action-space required to answer questions of locality is reduced to traversals and basic interactions with containers. Therefore, less exploration is needed as the information to answer a question is more easily acquired. Too high an initial reward would therefore promote unnecessary actions that would stand a high likelihood of leading the agent astray from stopping in the correct state.

6.1.2 Existence Existence questions require far more exploration of an environment than location type questions. Higher starting rewards reflect this need for greater exploration and are associated with better QA and sufficient information scores, as seen in Table 4. In turn, these higher values promote a more complete traversal of the world, allowing for gathering information required to answer the question. However, too high an initial reward means that entering a correct state and receiving a reward of 1.0 is unlikely to affect the model's decision making. If the DT has a current RTG of 5.0 and enters the correct state that rewards 1.0, the RTG from then onwards is 4.0. The return-to-go of 4.0 does not indicate to the model that it has entered the correct state, meaning it carries on exploring and gathering information. Likewise, too small a reward could prematurely cause an agent to stop exploring due to gaining rewards for entering new states via the exploration bonus. Therefore, we observe that the RTG value errs on the larger side, relative to all values tested, which we posit encourages greater world exploration. Conversely, this value cannot be too large as it needs to factor in that a significant deduction from future reward will affect decision-making.

6.1.3 Attribute Attribute type questions are considered the most sparsely rewarded of all three types [26]. We would therefore expect higher rewards to be associated with better scores. The results, however, paint a slightly different picture. In a fixed map, where the state-space is, on average, smaller than that of random, we see that a smaller reward yields the best score. This reduction is likely a result of the fixed map environment having a reduced state and action space are smaller than that of random, making too much exploration and interaction with the environment degrade performance. On the other hand, in a random map setting, higher rewards yielded better QA and sufficient information scores, allowing us to conclude that higher reward promotes more exploration and thus allows the model to answer the question better.

6.2 DT vs DT-BERT for QA

At a high level, the performance differences between the BERT and DT for question-answering depends on the state and action space that the model was required to learn and navigate. To this end, we found the Decision Transformer to have outperformed the BERT model with respect to questions that require the modelling of long-lasting dependencies. Such questions whose answers were more likely to be found within the last 512 tokens saw DT-BERT achieve higher question-answering accuracy than the Decision Transformer. Therefore, DT-BERT outperformed the Decision Transformer's answer prediction capabilities in location and attribute type questions, with the DT showing a higher accuracy on existence questions.

6.2.1 Location We observed location type questions to be more accurately answered on the test set when using the BERT QA module instead of the Decision Transformer's prediction head. The reason for this result is likely twofold. First, the BERT model learns skills akin to basic pattern matching when answering location questions. Thus, identifying entity and location names from state strings when the DT predicts to stop is likely trivial to learn. Second, exploration is not as highly encouraged with location questions as with existence and attribute types. Less exploration means fewer states visited, allowing the state context window to contain less noisy state strings than other question types. We also posit that this extra context allows

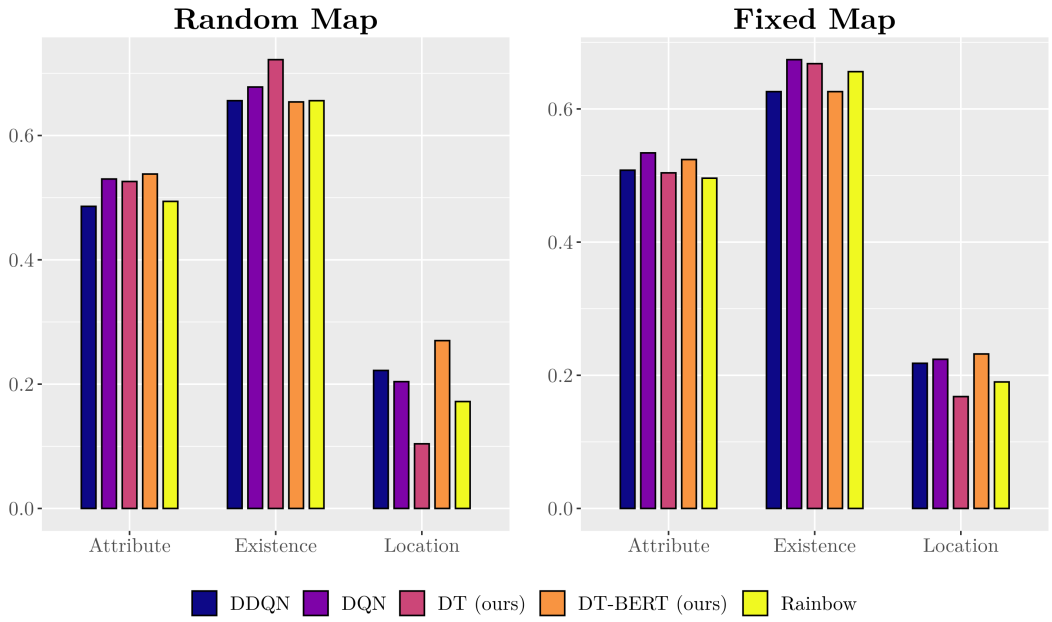

Figure 4: Barplot showing QA accuracy of the Decision Transformer's answer-prediction head and the BERT QA model compared to the DQN, DDQN, and Rainbow agents when trained in the 500 games setting. See results in Tables 6 and 7.

the BERT model to achieve a QA accuracy higher than the sufficient information score.

6.2.2 Existence Reasoning about the existence of an object within a TextWorld environment requires knowledge about the entirety of the world. Therefore, existential questions require an agent to fully explore an environment to answer whether or not an entity exists within it. As the Decision Transformer utilises the GPT-2 architecture, GPT-2's self-attention mechanism makes performing long-term credit assignments on a particular state possible. The answer-prediction head of the DT can thus draw upon information gathered in all previous states to inform decision making and question-answering. As a result, the ability to model dependencies that stretch throughout all states encountered allows the DT to outperform the BERT model, whose context window is constrained to 512 tokens.

6.2.3 Attribute The Decision Transformer failed to outperform the question-answering capabilities of the DT-BERT combination on zero-shot evaluation with attribute type questions in both fixed and random map types. Despite the Decision Transformer's ability to learn long-term dependencies via GPT-2's attention mechanism, we posit that the language embeddings of the BERT model were able to model a richer semantic representation of TextWorld's state-strings than the embeddings learnt by the DT. This better capturing of the semantic space meant that BERT could more fully utilise the context with which it was provided by using pre-existing understanding to help answer questions posed in natural language.

6.3 Sample Efficiency

The number of offline trajectories used to train the Decision Transformer is available in Table 1. The RL agents and QA module in QAit were trained for 200K episodes. In comparison, the resulting sufficient information and question-answering accuracies achieved on the test-set provide an indication of the DT's sample efficiency. That is, the DT is able to outperform the previous RL methods when trained on approximately 25% of the amount of episodes. Moreover, all training data used for the DT was generated via random rollouts - indicating that the Decision Transformer has the ability to learn optimal policies from suboptimal data, affirming both our initial hypothesis as well as results by Chen et al. [5]. We also found even when fine-tuning a BERT model for QA on the random rollout data, we managed to perform on par with the previous QA accuracies.

7 CONCLUSIONS

This paper shows that current reinforcement learning baselines set out in the QAit task can be matched and improved upon by framing IQA as a supervised sequence modelling problem and using a Transformer architecture for action generation and answer prediction. Additionally, we showed an improved sample efficiency when training, using a smaller training set than what reinforcement learning necessitates, consisting of suboptimal data generated via random rollouts. Moreover, when fine-tuning a BERT model on this same dataset for question-answering and allowing it to work in tandem with the Decision Transformer for action generation, several QAit QA baselines were outperformed.

8 FUTURE WORKS

For future works, we propose testing the limits of the Decision Transformer's sample efficiency and generalisability. This would require the DT be trained on extremely data deprived samples, consisting of fewer than 10K trajectories. Furthermore, we also suggest such

an approach makes little use of exploration rewards, instead relying purely on the Decision Transformer navigating to the correct state. This can help to better understand the DT's ability to assign long term credit in sparse-reward settings - testing the extent to which its state-of-the-art reward agnosticism can be utilised. Moreover, we propose a Longformer [2] architecture that allows for a far greater context window of up to 4096 tokens - testing the extent to which extra information can aid interactive question-answering. By increasing the context length, we believe QA performance would drastically improve. Lastly, we suggest fine-tuning a BERT model, or equivalent, to embed state and actions that are passed to the Decision Transformer's prediction heads in the hopes of utilising pre-existing language understanding to better generate actions and predict answers.

9 ACKNOWLEDGEMENTS

Thank you to Dr. Jan Buys for the invaluable guidance and assistance throughout the duration of the project. Thank you to Dr. Jonathan Shock for co-supervising and providing reinforcement learning insights and counsel.

References

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[2] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[5] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345* (2021).

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[7] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2018. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*. Springer, 41–75.

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

[9] Michael P Georgeff and Amy L Lansky. 1986. Procedural knowledge. *Proc. IEEE* 74, 10 (1986), 1383–1398.

[10] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. 2018. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4089–4098.

[11] P Hayes-Roth, M Fox, G Gill, DJ Mostow, and R Reddy. 1976. Speech understanding systems: Summary of results of the five-year research effort. (1976).

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Reinforcement Learning as One Big Sequence Modeling Problem. *arXiv preprint arXiv:2106.02039* (2021).

[14] Dan Jurafsky and James H Martin. 2014. Speech and language processing. Vol. 3. *US: Prentice Hall* (2014).

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Thomas Mesnard, Théophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Tom Stepleton, Nicolas Heess, Arthur Guez, et al. 2020. Counterfactual credit assignment in model-free reinforcement learning. *arXiv preprint arXiv:2011.09464* (2020).

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 1310–1318.

[18] Tatiana-Andreea Petrache, Traian Rebedea, and Ştefan Trăuşan-Matu. [n.d.]. Interactive language learning-How to explore complex environments using natural language? ([n. d.]).

[19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[20] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092* (2021).

[21] Saku Sugawara, Kentaro Inui, Satoshi Sekine, and Akiko Aizawa. 2018. What makes reading comprehension questions easier? *arXiv preprint arXiv:1808.09384* (2018).

[22] M Onat Topal, Anil Bas, and Imke van Heerden. 2021. Exploring transformers in natural language generation: Gpt, bert, and xlnet. *arXiv preprint arXiv:2102.08036* (2021).

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[24] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[25] Ziang Xie. 2017. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534* (2017).

[26] Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Christopher Pal, Yoshua Bengio, and Adam Trischler. 2019. Interactive language learning by question answering. *arXiv preprint arXiv:1908.10909* (2019).

[27] Xingdi Yuan, Jie Fu, Marc-Alexandre Cote, Yi Tay, Christopher Pal, and Adam Trischler. 2019. Interactive machine comprehension with information seeking agents. *arXiv preprint arXiv:1908.10449* (2019).

A Hyperparameter Tuning

A.1 Decision Transformer

A.1.1 Location As can be seen in Table 5, the sufficient information score peaking at $R_1 = 2$ indicates optimal state-space exploration for location questions when the potential for future reward is moderate for random and fixed map types. While the QA accuracy was highest for both settings when the initial reward was the maximum of the training set, we opted to test the DT's question answering and information gathering capabilities at $R_1 = 2$ as this yielded the highest combined sufficient information and QA accuracy score.

A.1.2 Existence Using both sufficient information and QA accuracy, the optimal initial reward for fixed map existence questions was determined to be 4.0, with the DT achieving a QA accuracy of 0.660 and a sufficient information score of 0.263 on the validation set. In random map settings, the DT scored a validation accuracy of 0.720 with a corresponding sufficient information score of 0.298, where the initial reward was determined to be the maximum of the training set 3.94.

A.1.3 Attribute The best sufficient information and QA accuracy combinations for the Decision Transformer were achieved at an initial reward of 2.0 for fixed and 5.0 for random map types. On the validation set, the fixed map DT achieved a QA accuracy of 0.533 and a SI score of 0.056. Random map saw a similar SI of 0.057 but worse QA accuracy of 0.460.

A.2 BERT Model

A.2.1 Location Based on data gathered using the online-evaluation dataset, the optimal initial return-to-go for location type questions was 2.0. Using the BERT model for QA yielded an accuracy of 0.227 for fixed maps and 0.393 for random maps. The BERT model

achieved a higher QA accuracy than sufficient information score during evaluation, indicating that the context window spanning multiple states was a boon to QA accuracy. During training, the BERT model achieved almost perfect scores for question-answering on the held-out set of offline trajectories, seen in Table 4.

A.2.2 Existence Using the online-validation set, we determined optimal starting reward values of 3.0 for fixed map and 3.94 for random. These values were associated with a QA accuracy of 0.64 for fixed and 0.647 for random map types. However, scores were significantly lower than the offline validation set used during training, where QA accuracy of 0.778 and 0.779 was achieved for fixed and random maps, respectively.

A.2.3 Attribute In the offline validation set, the BERT model scored a QA accuracy of 0.616 for random and 0.780 for fixed map settings. On the online validation set, we observed the maximum combination of QA and sufficient information for the BERT model at an R_1 of 3.0 for fixed and 2.0 for random where the BERT QA model had an accuracy of 0.507 and 0.660 for random and fixed map types, respectively. However, we opted to use the maximum of the train set 4.03 when evaluating on the test set for random map types. This is due to the BERT QA model having a high standard deviation of 0.156 and an average QA accuracy of 0.640, indicating greater potential for high QA accuracy. Moreover, the sufficient information score associated with this accuracy is 0.056 - higher than the random map with an initial reward of 2.0.

Table 5: Question-answering accuracy of BERT model and Decision Transformer's answer prediction head as well as Decision Transformer's average sufficient information (SI) score on validation set at different initial return-to-go (RTG) values. Bold values indicate the combined highest QA and sufficient information score with the associated initial RTG value also bolded. *Sampling* indicates R_1 was randomly sampled from an exponential distribution. *Max* represents the maximum of the training set for that experiment configuration (see Table 1). Summary statistics were calculated over 4 seeds - see code implementation for details.

Question Type						
Attribute						
	Fixed			**Random**		
Initial RTG	**BERT**	**DT**	**SI**	**BERT**	**DT**	**SI**
1	0.427 ± 0.0416	0.493 ± 0.0306	0.052 ± 0.0135	0.567 ± 0.0306	0.433 ± 0.0231	0.050 ± 0.0083
2	0.473 ± 0.0503	**0.533 ± 0.0115**	**0.056 ± 0.0094**	**0.660 ± 0.0200**	0.453 ± 0.0306	**0.048 ± 0.0039**
3	0.367 ± 0.0503	0.480 ± 0.0400	0.051 ± 0.0043	0.620 ± 0.0529	0.447 ± 0.0503	0.054 ± 0.0034
4	**0.507 ± 0.0643**	0.480 ± 0.0529	**0.056 ± 0.0058**	0.560 ± 0.1058	0.400 ± 0.0693	0.054 ± 0.0014
5	0.487 ± 0.0503	0.500 ± 0.0200	0.056 ± 0.0007	0.620 ± 0.0000	**0.460 ± 0.0346**	**0.057 ± 0.0033**
Sampling	0.487 ± 0.0416	0.453 ± 0.0231	0.051 ± 0.0050	0.593 ± 0.0503	0.433 ± 0.0231	0.052 ± 0.0097
Max	0.487 ± 0.0503	0.493 ± 0.0643	0.055 ± 0.0021	**0.640 ± 0.1562**	0.440 ± 0.0200	**0.056 ± 0.0020**
Existence						
	Fixed			**Random**		
Initial RTG	**BERT**	**DT**	**SI**	**BERT**	**DT**	**SI**
1	0.600 ± 0.0721	0.640 ± 0.0200	0.216 ± 0.0314	0.680 ± 0.0917	0.747 ± 0.0115	0.200 ± 0.0416
2	0.533 ± 0.0702	0.660 ± 0.0346	0.259 ± 0.0051	0.653 ± 0.0611	0.687 ± 0.0306	0.277 ± 0.0441
3	**0.640 ± 0.0721**	0.647 ± 0.0115	**0.265 ± 0.0161**	0.607 ± 0.0306	0.733 ± 0.0115	0.269 ± 0.0075
4	0.580 ± 0.0529	**0.660 ± 0.0400**	**0.263 ± 0.0180**	0.633 ± 0.0577	0.707 ± 0.0306	0.291 ± 0.0476
5	0.633 ± 0.0702	0.680 ± 0.0200	0.222 ± 0.0166	0.620 ± 0.0721	0.700 ± 0.0200	0.310 ± 0.0205
Sampling	0.613 ± 0.0416	0.647 ± 0.0231	0.180 ± 0.0467	0.653 ± 0.0416	0.707 ± 0.0416	0.250 ± 0.0070
Max	0.587 ± 0.0306	0.640 ± 0.0200	0.270 ± 0.0409	**0.647 ± 0.0115**	**0.720 ± 0.0400**	**0.298 ± 0.0302**
Location						
	Fixed			**Random**		
Initial RTG	**BERT**	**DT**	**SI**	**BERT**	**DT**	**SI**
1	0.165 ± 0.0191	0.175 ± 0.0379	0.165 ± 0.0191	0.267 ± 0.0306	0.107 ± 0.0115	0.267 ± 0.0306
2	**0.227 ± 0.0231**	**0.167 ± 0.0115**	**0.233 ± 0.0115**	**0.393 ± 0.0416**	**0.080 ± 0.0200**	**0.387 ± 0.0503**
3	0.187 ± 0.0416	0.167 ± 0.0306	0.187 ± 0.0416	0.307 ± 0.0306	0.120 ± 0.0200	0.307 ± 0.0306
4	0.173 ± 0.0115	0.133 ± 0.0231	0.173 ± 0.0115	0.360 ± 0.0400	0.093 ± 0.0115	0.347 ± 0.0306
5	0.167 ± 0.0462	0.160 ± 0.0200	0.167 ± 0.0462	0.340 ± 0.0346	0.080 ± 0.0346	0.333 ± 0.0306
Sampling	0.160 ± 0.0200	0.167 ± 0.0416	0.167 ± 0.0115	0.287 ± 0.0757	0.107 ± 0.0306	0.287 ± 0.0757
Max	0.193 ± 0.0231	0.187 ± 0.0115	0.193 ± 0.0231	0.333 ± 0.0945	0.120 ± 0.0200	0.327 ± 0.0702

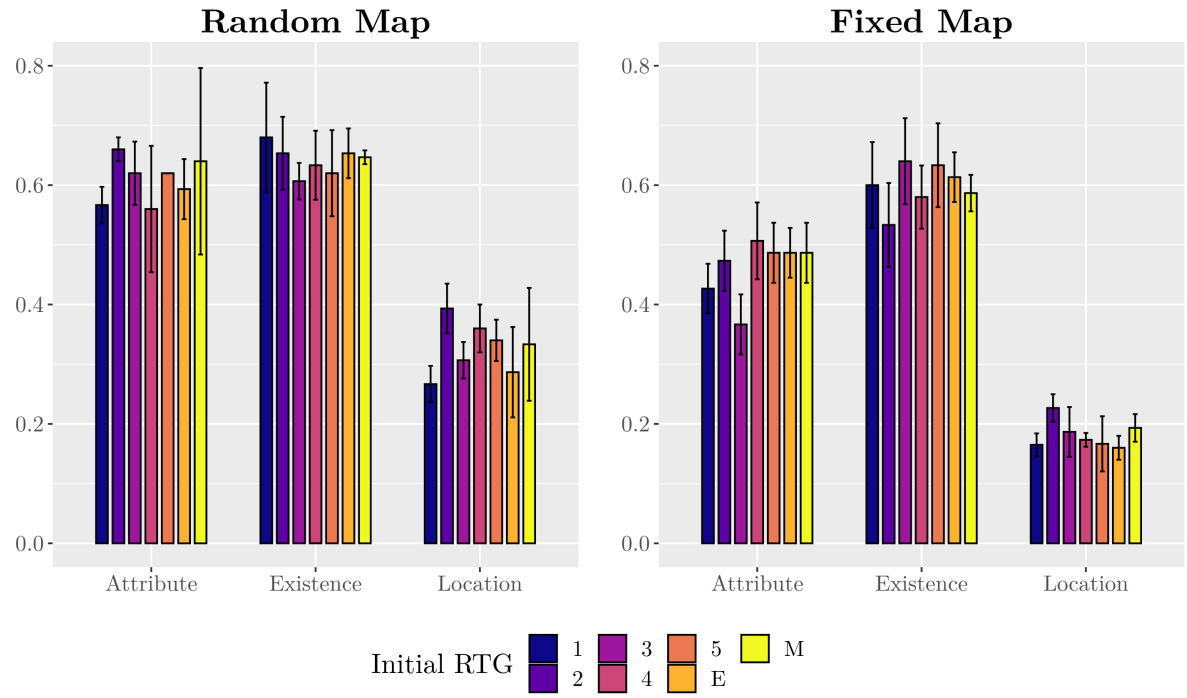

Figure 5: Barplot showing QA accuracy of the BERT QA model on the validation set when trained in the 500 games setting with different initial returns-to-go (RTG). See results in Table 5.

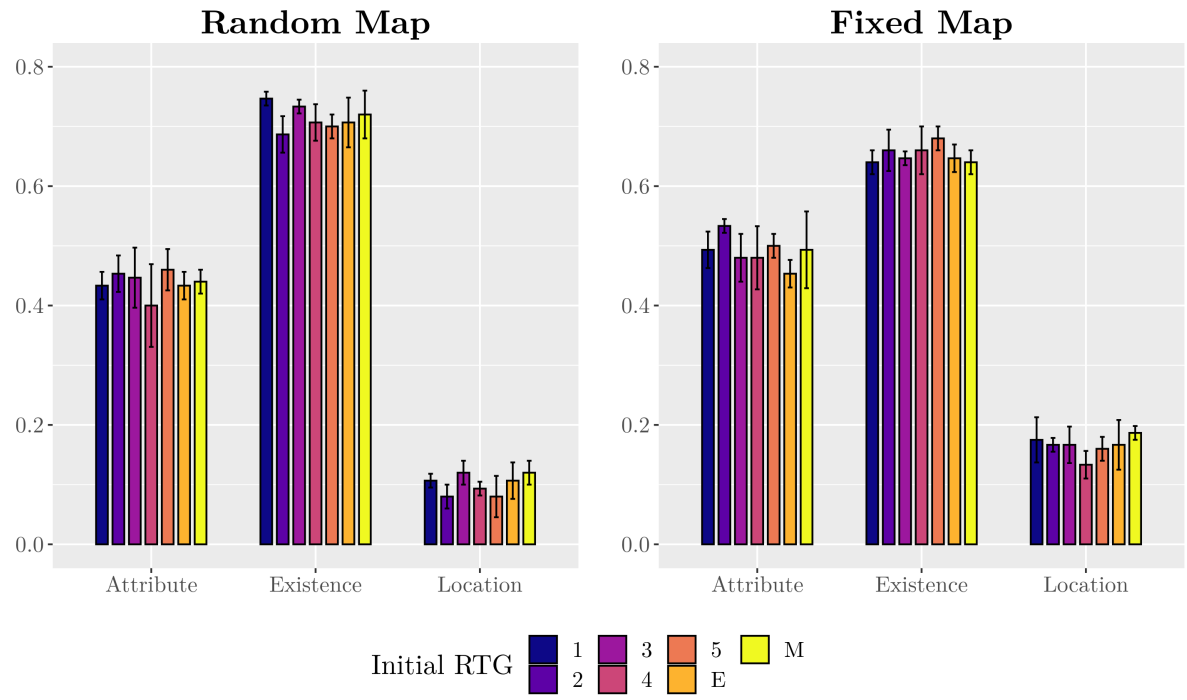

Figure 6: Barplot showing QA accuracy of the Decision Transformer's answer-prediction head on the validation set when trained in the 500 games setting with different initial returns-to-go (RTG). See results in Table 5.

Code Listing 1: Pseudocode showing the training and evaluation of the Decision Transformer for the QAit task.

```
# R, s, a, t: returns-to-go, states , actions , or  timesteps
# transformer: transformer  with  causal  masking (GPT)
# embed_R: linear  embedding  layer
# embed_words: linear layer embedding for both states & actions
# embed_t: learned  episode  positional  embedding
# encoder: GRU encoder for embedded states & actions

# pred_a: linear  action  prediction  layer
# pred_m: linear  modifier  prediction  layer
# pred_o: linear  object  prediction  layer
# pred_ans: linear  answer  prediction  layer

# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t) # per timestep, not per token

    # states and actions are first embedded and then encoded with a GRU
    s_embedding = encoder(embed_words(s)).last_hidden_state + pos_embedding
    a_embedding = encoder(embed_words(a)).last_hidden_state + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K, a_k)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # input interleaved embeddings and get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for command prediction
    s_hidden = unstack(hidden_states).states

    # predict answer to question and action, modifier, object of command  given state
    action = pred_a(s_hidden)
    modifier = pred_o(s_hidden)
    object = pred_a(s_hidden)

    answer = pred_ans(s_hidden)

    return action, modifier, object, answer

# training loop
for (R, s, a, t) in dataloader:
    a_preds = DecisionTransfomer(R, s, a, t)
    loss = cross_entropy(a_preds, a)
    optimizer.zero_grad(); loss.backward(); optimizer.step()

# evalutation loop
initial_reward = 1 # sampled from an exponential distribution or fixed
env = load_textworld_env()
R, s, a, t, done = [initial_reward], [env.reset()], [], [1],  False

while not done:

    act, mod, obj, _ = DecisionTransformer(R, s, a, t)[-1]
    command =  act + mod + obj # join action, modifier, and object together
    new_s, r, done = env.step(command)

    R = R + [R[-1] - r] # decrement return-to-go with reward
    s, a, t = s + [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K]
```

Table 6: Results of Fixed Map Experiments

Fixed						
Model	**Location**		**Existence**		**Attribute**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Random	-	0.027	-	0.497	-	0.496
500 games						
DQN	0.430 (0.430)	0.224 (0.244)	0.742 (0.136)	0.674 (0.279)	0.700 (0.015)	**0.534 (0.014)**
DDQN	0.406 (0.406)	0.218 (0.228)	0.734 (0.173)	0.626 (0.213)	0.714 (0.021)	0.508 (0.026)
Rainbow	0.358 (0.358)	0.190 (0.196)	0.768 (0.187)	0.656 (0.207)	0.736 (0.032)	0.496 (0.029)
DT	-	0.168 (0.232)	-	0.668 (0.254)	-	0.504 (0.057)
DT-BERT	-	**0.232 (0.232)**	-	0.626 (0.258)	-	0.524 (0.058)

Table 7: Results of Random Map Experiments

Random						
Model	**Location**		**Existence**		**Attribute**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Random	-	0.034	-	0.5	-	0.499
500 games						
DQN	0.430 (0.430)	0.204 (0.216)	0.752 (0.162)	0.678 (0.214)	0.678 (0.019)	0.530 (0.017)
DDQN	0.458 (0.458)	0.222 (0.246)	0.754 (0.158)	0.656 (0.188)	0.716 (0.024)	0.486 (0.023)
Rainbow	0.370 (0.370)	0.172 (0.178)	0.748 (0.275)	0.678 (0.191)	0.636 (0.020)	0.494 (0.017)
DT	-	0.104 (0.264)	-	**0.722 (0.277)**	-	0.526 (0.058)
DT-BERT	-	**0.270 (0.264)**	-	0.654 (0.277)	-	0.538 (0.060)

Table 8: Heuristic conditions for determining whether the agent has enough information to answer a given attribute question. We use "object" to refer to the object mentioned in the question. Words in italics represent placeholders that can be replaced by any object from the environment that has the appropriate attribute (e.g. carrot could be used as a cuttable). Pass and Fail columns represent how much reward the agent will receive given the corresponding command's outcome (resp. success or failure). [26]

Attribute	Command	State	Pass	Fail	Explanation
sharp	cut cuttable	holding (cuttable) & uncut (cuttable) & holding (object)	1	1	Trying to cut something cuttable that hasn't been cut yet while holding the object.
	take object	reachable(object)	0	1	Sharp objects should be portable.
cuttable	cut object	holding (object) & holding (sharp)	1	1	Trying to cut the object while holding something sharp.
	take object	reachable (object)	0	1	Cuttable object should be portable.
edible	eat object	holding (object)	1	1	Trying to eat the object.
	take object	reachable (object)	0	1	Edible objects should be portable.
drinkable	drink object	holding (object)	1	1	Trying to drink the object.
	take object	reachable (object)	0	1	Drinkable objects should be portable.
holder	-	on (portable, object)	1	0	Observing object(s) on a supporter.
		in (portable, object)	1	0	Observing object(s) inside a container.
	take object	reachable (object)	1	0	Holder objects should not be portable.
portable	-	holding (object)	1	0	Holding the object means it is portable.
	take object	reachable (object)	1	1	Portable objects can be taken.
heat_source	cook cookable	holding (cookable) & uncooked (cookable) & reachable (object)	1	1	Trying to cook something cookable that hasn't been cooked yet while being next to the object.
	take object	reachable (object)	1	0	Heat source objects should not be portable.
cookable	cook object	holding (object) & reachable (heat_source)	1	1	Trying to cook the object while being next to a heat source.
	take object	reachable(object)	0	1	Cookable objects should be portable.
openable	open object	reachable (object) & closed (object)	1	1	Trying to open the closed object.
	close object	reachable (object) & open (object)	1	1	Trying to close the open object.