The Role of Graph Attention Networks in Interactive Question Answering Systems

Roy Cohen
CHNROY002@myuct.ac.za
UCT Computer Science Honours

ABSTRACT

The ability for models to achieve humanlike language comprehension abilities has been a long aspired goal of artificial intelligence and many natural language processing tasks. Interactive Question Answering (IQA) has been proposed to address the lack of understanding and comprehension abilities of current systems. To this end, the Question Answering with interactive text (QAit) task was created using text-based environments and provides a framework for training and evaluating a models ability to perform IQA and thus displaying its knowledge gathering capabilities. This paper aims to investigate the use of Graph Attention Networks in providing additional context to agents to increase their question-answering abilities. Results indicate that Graph Attention Networks can increase the question answering accuracy of an agent when training on data containing high variability, thus improving the agent's generalisation abilities.

KEYWORDS

Question Answering Systems, Interactive Question Answering, Graph Attention Network, Graph Neural Network, Natural Language Processing, Reinforcement Learning, Text-based Games.

1 INTRODUCTION

Machine reading comprehension's (MRC) primary focus is on the retrieval of explicitly stated knowledge or static descriptions of entities, known as *declarative knowledge*, within text or knowledge-bases (KBs) [48]. Evidence suggests that current MRC and neural methods find it challenging to acquire actual *comprehension* and *understanding* that is deemed necessary in many Natural Language Processing tasks, such as question-answering [41, 48, 50].

Simple pattern matching of natural language as a method of extracting knowledge from a knowledge source is preponderance when training models on current MRC datasets [49]. Current models lack essential humanlike skills needed to interact with and observe environments to gain understanding about it. Simple pattern recognition does not reward behaviour necessary to answer many natural language questions, such as information-seeking [32]. To promote information-seeking behaviours, *procedural knowledge* rewards a model for gathering the declarative knowledge required to answer a question. Procedural knowledge thus shifts the focus from being less on answering the question or completing a task but rather on the actions performed to complete the task.

A proposed solution to the lack of understanding and comprehension of current methods is Interactive Question Answering (IQA). IQA is a combination of question answering systems and dialogue systems, where question answering requires a system to ask questions in natural language about some dynamic environment and receive answers from the model. In contrast, dialogue systems enable the model to exchange in dialogue with the system in cases where there are multiple answers, no answers, ambiguity, or the model needs to take actions to arrive at the solution [21, 31]. The aforementioned environment can take many forms, such as a KB [18], some entity-relation schema or a text-based game [14, 48].

Text-based games are slowly becoming a *de facto* standard for dynamic environments, as it allows for language-learning problems to be tackled using various reinforcement learning (RL) methods in dialogue-like settings [28, 37]. One of the more popular text-based game frameworks is Microsoft *TextWorld* [14, 37], which was used by Yuan et al. to develop the Question Answering with Interactive Text (QAit) task [48]. The task allows an agent to interact with a partially observable text-based environment to answer questions in a *procedural knowledge setting* about the location, existence and attributes of objects in the environment. To complete the task, the agent must successfully gather and seek out information to answer questions.

Within the context of TextWorld, however, many agents fail to generalise and capture necessary meaning and relationships between entities found in the environment [49]. A model failing to develop some contextual awareness of the world can be detrimental to performance. Current works indicate knowledge graphs (KGs) help express supplementary information about the world to facilitate decision-making better whilst adhering to partial observability [2–4].

This paper explores the use of Graph Attention Networks in providing an RL agent with some contextual understanding about the environment with which it inhabits, which has shown to aid aspects of performance such as steps required to complete a task and training convergence [3]. This additional context results in action sequences better suited to goal achievement by providing circumstantial information regarding the action space. Therefore, by embedding specific details about an environment into a knowledge graph and allowing an agent to use it to inform decision-making, this paper explores the effects of Graph Attention Networks on agents' accuracy on the QAit task [49].

More specifically, this paper aims to answer the following research questions:

(1) Do Graph Attention Networks increase an RL agent's question-answering capabilities?
(2) Can Graph Attention Networks speed up the training convergence of an agent in an IQA system?

2 BACKGROUND

2.1 Knowledge Bases

Knowledge Bases (KBs) are frequently used as a structured database that stores data in the form of tuples. Tuples are in the form of `<head, relation, tail>` [10, 19, 47] and natural language questions are typically translated into a KB query in the form of

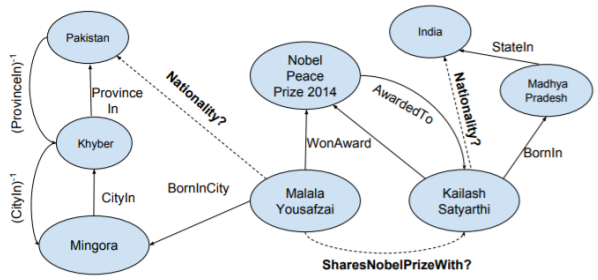

Figure 1: A small piece of a KB represented as a KG, from Das et al. "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning" [16]. Solid edges are KB relations and dotted lines are queries.

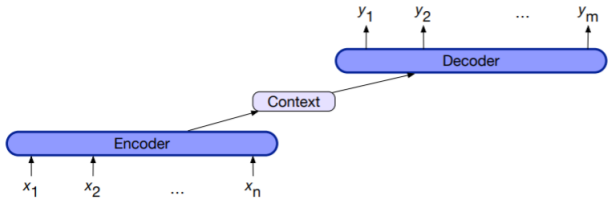

Figure 2: Sequence-to-sequence model architecture, from Jurafsky "Speech and language processing" [27].

<head, relation, ?>. An example of this would be <Cape Town, cityIn, South Africa> where the tail entity would be the answer to the question.

Since the introduction of free large-scale tuple based KBs, such as *Freebase* [9], and *DBPedia* [6], they have been extensively used as an open-domain source of information for many different Natural Language Processing tasks such as entity-linking, relation extraction, and question answering systems [29, 40].

A *Knowledge Graph* (KG) is a graph-based representation of a KB created by treating *entities* as nodes and *relations* as edges, which is illustrated in Figure 1. KGs can express context by storing necessary information relating to a task and, more importantly, storing information relating to a task's past events, which can aid in the long-term memory of agents or models. Using KGs to store tuples related to the context of a problem has demonstrated the ability to improve natural language understanding in Natural Language Processing tasks both inside and outside of text adventure games [2–4, 23].

2.2 Graph Neural Networks

Many tasks such as image classification, semantic segmentation or machine translation have an underlying data representation of a grid-like structure [44] and thus have been successfully modelled using Convolutional Neural Networks (CNNs) [20, 24, 25]. However, many tasks involve data that cannot be represented in such a structure but rather in an irregular domain that is representable as graphs (e.g. 3D meshes, telecommunication networks) [44].

Graph Neural Networks (GNNs) [22, 39] were developed to deal more generally with different classes of graphs such as cyclic, directed or undirected through the use of recursive neural networks. The iterative process of GNNs produces an output for each node in the graph based on its state; this is done by propagating the node states until equilibrium, then inputting the node representation into a neural network. GNNs were later improved through the use of Gated Recurrent Units (GRUs) [13] when propagating node states [33]. GNNs effectively transform graphs, such as KGs, into different representations that fit the domain on which they are trained.

More recently, the idea of attention mechanisms was incorporated in GNNs to produce the Graph Attention Network (GAT) [44], managing to achieve or match state-of-the-art results. The GAT computes the representations of each node in the graph by attending over its neighbours, following a *self-attention* strategy (see Section 4.3).

2.3 Sequence-to-sequence Models

Sequence-to-sequence models, are models that manage to compute a contextual output sequence that isn't limited by output length. Many applications exist for sequence-to-sequence models such as semantic parsing [35], syntactic parsing [45], image captioning [46], machine translation [13, 30], etc.

The underlying architecture is to have an encoder network that translates natural language into a contextualisation representation of itself (the *context vector*) [27]. The decoder then processes the context vector, which outputs a sequence of tokens. Figure 2 shows a visual representation of the sequence-to-sequence model.

The use of *Recurrent Neural Networks* (RNNs) [13] as the encoder and decoder in a sequence-to-sequence model allows the model to process variable-length inputs and outputs [34]. RNNs are neural networks that contain cycles [27]. Thus the encoder-decoder RNNs are trained together to learn to encode any length natural language input into a set length context vector and decoded back into a sequence of natural language tokens [34].

Sequence-to-sequence encoders' ability to encode variable-length input allows components with variably sized outputs (such as KGs and GNNs) to be transformed into a fixed-length representation. Fixing the output size of a component that otherwise is variable in length makes for easier integration of the component with the rest of the architecture.

2.3.1 Transformers. Transformers [43] are sequence-to-sequence models that approach the sequence processing task by eliminating recurrent connections [26]. Transformer encoders are comprised of a self-attention layer, feedforward neural network, a normalization layer [7] along with custom connections around them. The key difference between transformers and other sequence processing models is its use of self-attention layers.

2.3.2 BERT. Bidirectional Encoder Representations from Transformers (BERT) [17] is a Transformer-based pre-trained model that achieves state-of-the-art performance when fine-tuned to create models for a wide range of tasks. Without needing to fine-tune BERT, it can act as a state-of-the-art word embedder, which translates natural language to vector representations [11].

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns how to behave by mapping situations in an environment to actions so that it can maximise a reward signal it receives for performing these actions [42].

Reinforcement Learning frames the decision-making problem into a *Markov Decision Process* (MDP) [8, 38], a mathematics-based method for solving decision-making problems with four key elements.

2.4.1 Elements of MDP. A *policy* dictates the agent's way of making decisions by mapping a set of observed states in the environment to actions it will perform [42]. Generally, policies can be stochastic by providing a probability for each action.

A *reward signal* defines a reinforcement problem's goal by providing a reward to the agent for making decisions (i.e. actions which will lead the agent closer to its goal will have more rewards than ones that do not). The agent's objective is to make decisions that will maximise its rewards. Furthermore, it is the reward function that will motivate a change in an agents policy. If a policy's action leads to low rewards, then the policy may be altered [42].

While the reward signal indicates immediate rewards for taken actions, the *value function* specifies what action is good in the long run [42]. In other words, the total amount of reward expected from the agent over multiple time steps. The agent must learn the value function to maximise the long term effect of taking an action by considering the available actions and rewards in the next state [42].

The last element is the *model*, which allows for inferences about what is accessible to the agent regarding actions in the environment—effectively allowing an agent plan for future states and their rewards. As this element is not always available to the agent, RL algorithms that do not use a model are *model-free*, whereas ones with models are *model-based* [42].

2.5 TextWorld

In 2019, Côté et al. [15], introduced TextWorld, which is a Python library that provides an interactive text-based partially-observable environment for Reinforcement Learning agents to train and learn various skills such as decision making and language comprehension. More specifically, it creates a framework for interactive text-based games to be developed. As a text-based game, TextWorld can be used to research and develop more generalised QA systems [1].

The interactive environment created is based on a provided or randomly generated KB, where entities and relations translate to locations and actions in TextWorld, allowing agents to roam the environment and maximise their rewards by learning the optimal policy.

3 RELATED WORK

3.1 Interactive Question Answering

In 2019, Yuan et al. [49] developed the Question Answering with Interactive Text (QAit) task, which introduces a text-based question answering task. To complete the task, agents must interact with a TextWorld environment and gather information required to answer given questions, as seen in Figure 3. The purpose of the task is to develop agents with greater generalisability and language understanding. The task also proposes and evaluates a set of deep reinforcement learning baseline agents for future research to evaluate against their agents.

Game	You find yourself in a backyard. You make out a patio table. You see a patio chair. It is stylish. In a corner you can see a BBQ. There is an open screen door leading south.
Question	Is ghargh edible?
Agent	> **Qapla'**
Game	That's not a verb I recognize.
Agent	> **eat ghargh**
Game	You can't see any such thing.
Agent	> **go south**
Game	You've just entered a kitchen. You can see an open fridge here. It is empty! You see a closed oven. Look over there, it's a counter. On the counter you can see a diced potato, a red ghargh and a knife. There is an open screen door leading north.
Agent	> **eat ghargh**
Game	You need to take the red ghargh first.
Agent	> **take red ghargh**
Game	You take the red ghargh from the counter.
Agent	> **eat ghargh**
Game	You eat the red ghargh. Not bad.
Agent	> **wait**
Agent	> **answer: yes**

Figure 3: An agent interacting with a text-based environment in order to gather necessary information to answer a question (example from [49]).

The task includes three baseline agents to evaluate against, namely the DQN, DDQN and Rainbow, some of which are the current best performing models on the QAit task. The baseline agent used in this paper, the DQN [36], consists of four main components: the encoders, an aggregator, a command generator and a question answerer, as depicted in Figure 4. The encoders are responsible for getting vector representations of the environment description and the question at each time step, while the aggregator combines them into a single representation. In contrast, the command generator and the question answerer utilise the aggregated representation to perform action pruning and the answering of the given question once the agent stops exploring, respectively.

3.2 GAT based RL agents in Text-Based Games

In 2019, Ammanabrolu et al. [2] showed how constructing a KG based on the observation descriptions provided by a text-based environment along with a GAT allowed an RL agent to learn a control policy faster than alternative baseline agents of text-based games. Additionally, the GAT allowed the agent to finish the game in fewer steps than other baselines. However, despite these benefits, the GAT assisted agent had worse results in the text-based games than the alternative baselines and underperformed overall.

While the reason for this shortcoming is not made abundantly clear, the implementation failed to take full advantage of the variable

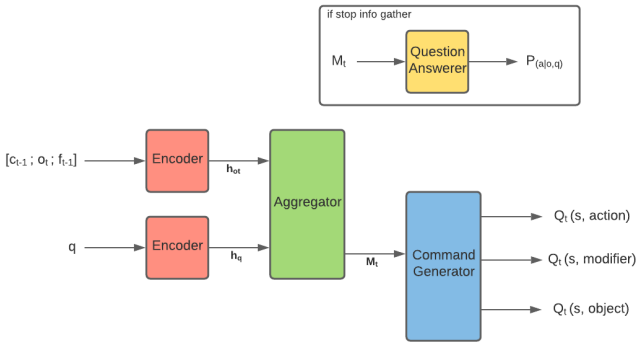

Figure 4: QAit task's baseline agent architecture.

sizing of GATs' inputs and outputs and instead kept the input into the GAT constant (i.e. same number of nodes) and only changing the relations (edges) connecting them. While this does not affect the GAT itself, it has broader implications on the system's performance and generalisability.

In order to have a fixed-sized input into the GAT, the system must know the total number of possible entities (nodes) the KG will encounter during the task. Meaning that a pre-processing agent was used to gather as many entities as possible from the training data before training commences, after which no new entities discovered were added to the KG during training (i.e. the KG will ignore any entities not discovered by the pre-processing agent). These implementation details would result in poor generalisability on unseen environments and likely explain the lack of comprehension abilities of the agent, which could have played a factor in the underwhelming performance of the model.

4 DESIGN AND IMPLEMENTATION

4.1 The QAit Task

4.1.1 Environment & Difficulty. As done by Yuan et al., Textworld will be used jointly with the QAit's world and question pair generator [49], which generates question-answer pairs related to TextWorld environments. QAit provides a framework for testing an agent's language comprehension abilities by asking various questions about an environment that require an understanding of locality, existence, and attributes. Environments have two different configurations, namely *fixed* and *random* maps. As shown in Table 1, fixed maps generate TextWorld environments with a fixed number and layout of rooms within the environment, whereas, with random maps, these numbers and layouts are sampled from a uniform random distribution.

The QAit task has fixed map and random map baseline agents for agents trained on 1, 2, 10, 100, 500 and *unlimited* games. Unlimited games entail generating games during training by randomly sampling the various environment parameters tabulated in Table 1.

Lastly, the QAit task provides two test sets that constitute 500 never-before-seen games, one for fixed maps and another for random maps. These games, held out during training and each containing a single question the agent must answer, were proposed by Yuan et al. to assess a model's generalisation abilities in a manner analogous to the test set of supervised learning problems.

Table 1: Distribution of locations and entities within QAit's generated environments for fixed and random maps. [49]

	Fixed Map	Random Map
# Locations, N_r	6	$N_r \sim Uniform(2, 12)$
# Entities, N_e	$N_e \sim Uniform(3 \cdot N_r, 6 \cdot N_r)$	

4.1.2 Question Types. The QAit task outlines three types of questions the agent will be attempting to answer based on the world generated:

- **Location**: Location type questions entail asking the agent to find the location of objects, such as "Where is the box of cereal" which would expect an answer like "pantry".
- **Existence**: These questions concern the existence of objects in the world, and an example of this question would be "Is there a scooter in the world" where the correct response would be a yes or no.
- **Attribute**: Attribute questions, deemed the most challenging, ask about the object's attributes, such as "Is the car red?" where the correct answer would be a yes or no. In order to avoid the agent simply memorizing attributes of objects, such as "Is a pizza edible", which would remain true between different games, objects are assigned random fictitious names.

4.2 Knowledge Graph Construction

In order for an agent to learn a KG, a set of RDF triples, i.e. a tuple of *(subject, relation, object)* are to be extracted and stored. The tuples are extracted from the observations provided by TextWorld at each agent step using Standford's Open Information Extraction (OpenIE) [5]. However, OpenIE was not designed with the regularities of Text-based adventure games in mind. To remedy this, Ammanabrolu et al. [2] outlined a set of heuristic-based rules to fill in the information not inferred by OpenIE. Combining the information extracted from OpenIE with the additional rules results in a KG that can provide an agent with representation of the game world.

Every time the agent acts, the KG updates to reflect the new information provided by the game. However, to provide the agent with both long term and short term context, a special node "you" is introduced to reflect what entities are currently applicable to the agent, as depicted in Figure 5. Using the "you" node, Ammanabrolu et al. [2] developed the following set of rules used to update the graph after each agent action:

- Create a link between the current room node (e.g. "bedroom") to nodes representing the items found in the room, using the relation "has" (e.g. <*bedroom, has, bed*>).
- Linking information regarding entrances and exits to the current room node (e.g. <*bedroom, has, exit to south*>).
- All relations relating to the "you" node are removed, except for the nodes representing an agent's current inventory in the game (e.g. <*you, have, key*>).
- Link the different room nodes with directions based on the agent's action taken to move between the rooms (e.g. <*bathroom, east of, bathroom*>).

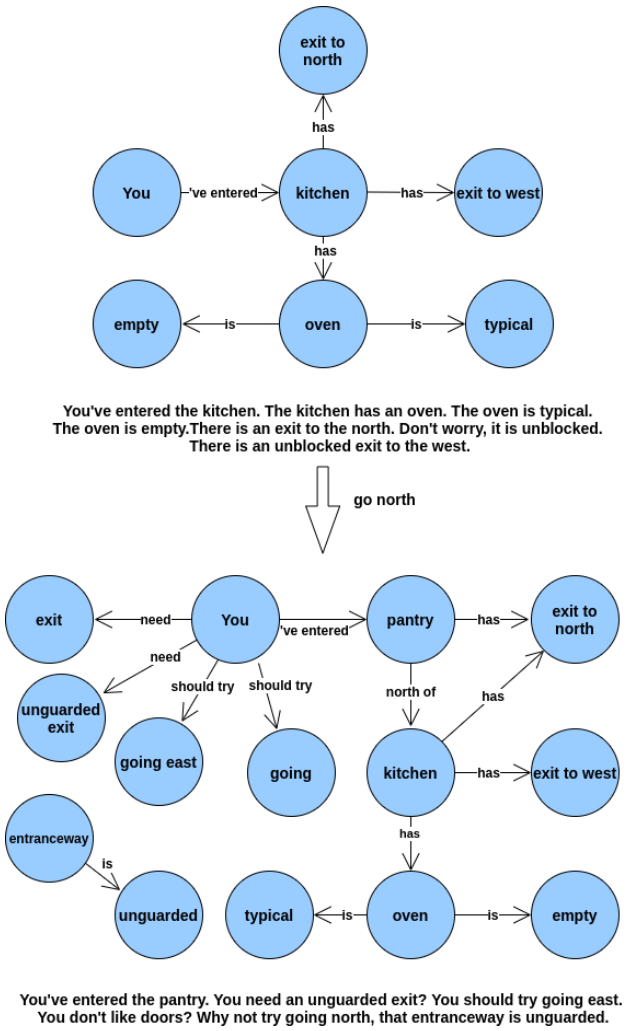

Figure 5: Knowledge Graph updating from one observation to the next given the agent action "go north".

OpenIE extracts all the other RDF triples added to the KG. Figure 5 shows an example of the KG updating given two observations and an action.

4.3 Graph Attention Network

The KG constructed at each agent action (see Section 4.2) provides the basis of the input into the GAT. The GAT's input is a set of node features, $\mathrm{h} = \{\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\}$, $h_i \in \mathbb{R}^F$, where N is the number of nodes and F is the number of features in each node. As such, each node in the constructed KG gets represented as a set of features. A pre-trained BERT model [17] obtains the set of features by embedding the natural language label of each node into a vector representation of size 512 (i.e. a node embedding).

4.3.1 Graph Attention Layer. In the Graph Attention Layer [44], the set of node features undergo a shared learnable linear transformation $\mathbf{W} \in \mathbb{R}^{F \times F}$. A *self-attention* mechanism is then applied which consists of weight vector $a \in \mathbb{R}^{2F}$ and the LeakyReLU nonlinearity (with negative input slope $\alpha = 0.2$):

$$e_{ij} = \text{LeakyReLU}(\vec{a}^T [\mathbf{W}\vec{h}_i || \mathbf{W}\vec{h}_j]) \quad (1)$$

where $.^T$ represents transposition and $||$ is the concatenation operation.

The graph structure is then injected into the process by performing *masked attention* - e_{ij} is only computed for nodes $j \in n_i$, where n_i is the first-order neighbours of node i in the graph. The *attention coefficients* are then computed by normalizing e_{ij} across all choices of j using the softmax function:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in n_i} \exp(e_{ik})} \quad (2)$$

After which, the normalized attention coefficients are used to compute a linear combination of the features corresponding to them, to produce the set of final output features for every node $\mathrm{h'} = \vec{h'}_1, \vec{h'}_2, ..., \vec{h'}_N$, $h'_i \in \mathbb{R}^F$ which is the GAT's output.

4.3.2 Variable Length Input/Output. Both the GAT's input and output are variable in length as the number of inputted nodes depend on the agent's phase in the game and how many entities are in the graph, which can vary from game to game and action to action.

In order to allow the agent to utilize the GAT's output when deciding which action to perform or how to answer a question, the GAT's output's length needs to be of fixed length to allow integration into QAit's baselines agent's architecture. A single layer trainable Transformer encoder [43] achieves this by taking the GAT's output as input to produce a fixed-length vector representation for the agent, effectively summarizing the transformed KG into a fixed-length vector representation.

4.4 Architecture

Figure 6 shows the complete process, at each time step t, of the implemented system and its core components. The KG Constructor takes in the observation provided by TextWorld at time step t (denoted as O_t), the agent's previous action C_{t-1} and the extracted relations and entities from OpenIE. Utilising the three inputs, it uses the set of heuristic rules defined in Section 4.2 to construct a KG. The constructor then uses a BERT model to obtain a set of node features at each time step h_t. The GAT uses h_t along with the edge indices (i.e. each node's first order neighbours) from the constructed graph to produce the set of output features h'_t. The Transformer Encoder turns the produced node features, h'_t, into a fixed-length summary S_t. S_t is then utilised by the QAit's Command Generator when ranking actions to perform and by the Question Answerer when answering questions. The length of S_t is set to 64 in the experiments to mirror the size of the match representation (M_t in Figure 4), which is the other input into the Command Generator and Question Answerer, respectively.

5 EXPERIMENT DESIGN

5.1 Experiments

Due to time and computational resource constraints, only a subset of the total experiment settings offered by the QAit benchmark is

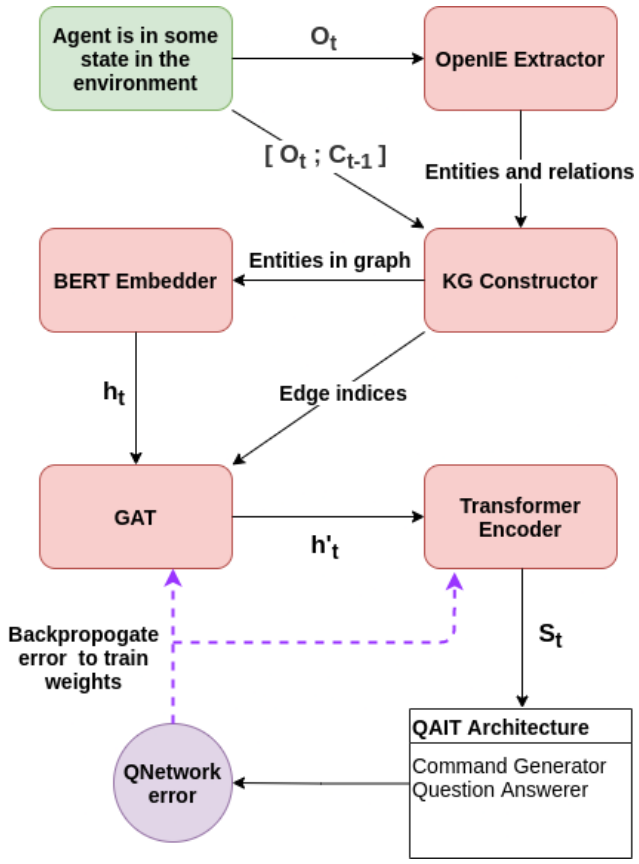

Figure 6: The overall process and architecture of the implemented system, including all inputs and outputs of the components required.

selected. There are six core experiments, with all six experiments trained on 500 different games with QAit's vanilla DQN agent. The DQN agent's simplicity compared to the other agents means it will have the least complexities or external factors to consider, which will maximise the paper's ability to infer the effect GATs have on an agent's question-answering ability when used as additional information. 500 games are chosen as the number of training games as it will not have too little training data, which could hinder the model's performance. Furthermore, 500 games is not an infinite amount, like unlimited games, which is not available for agents in most instances of IQA, allowing us to evaluate the effectiveness of GATs in IQA systems more generally.

The six experiments will be a combination of the type of questions asked (location, existence or attribute) and the type of game map (fixed or random maps). Fixed maps mean all the games in the training set will have the same number of rooms and thus less variability; conversely, random maps randomly sized each game. All experiments will train for 200 000 episodes as done by the baseline agents in the QAit task.

As time and computational constraints allowed for a few more experiments, three additional experiments were run on random maps with *unlimited* training games (instead of 500) to provide additional evidence for a postulate inferred from the core experiments run (see Section 7). Unlimited games entail games being generated during training with the various values for the environment parameters tabulated in Table 1 being randomly sampled, effectively ensuring that each training episode sees a different game.

5.2 Evaluation

An agent's accuracy will be measured using zero-shot evaluation on the QAit task's test sets and compared to the QAit baselines in the following metrics:

5.2.1 Question Answering Accuracy. Question Answering (QA) accuracy is the proportion of questions the agent managed to answer correctly. QA accuracy is calculated the same way regardless of question type or game configuration. The core function of QA systems is to answer questions; thus, this metric is argued to be the most critical indicator of the performance of a model.

Along with the QA accuracy, one could conduct other statistical performance measures such as precision, recall or the F1 score, allowing for detailed, in-depth explanations for what the agent is learning to answer should the deeper analysis be required.

5.2.2 Sufficient Information Score. The sufficient information (SI) score is a measure detailing the amount of gathered information by the agent [49]. It allows for examining whether agents got sufficient information to answer the question, which is an indicator to evaluate the exploration and actions needed to answer a given question correctly. Each of the three questions types calculate the SI score differently to best suit the type of interactions required by the agents to answer a question.

- **Location**: When the agent decides to stop interacting with the environment and provide the answer to the question, it will receive an SI score of 1 if the environment revealed the answer to the question in the last observation provided by the environment and a score of 0 otherwise. A score of 1 indicates that the agent recognises that the environment provided enough information to answer the question.
- **Existence**: The SI scores for existence questions are calculated differently based on the actual answer of the question:
  - Answer is yes: Similarly to location type questions, if the answer is yes, then a score of 1 is given if the entity in question is in the latest observation provided by the environment; otherwise, the agent receives a score of 0.
  - Answer is no: The agent will receive a score of between 0 and 1 depending on the proportion of exploration done by the agent before answering, as more exploration is needed to say an entity does not exist in the environment confidently.
- **Attribute**: To evaluate an agents SI score on attribute type questions, the QAit task developed a set of heuristics (outlined in Table 4 in the Appendix) allowing for specific SI scores to be assigned to different attributes.

5.2.3 Training Episodes. In order to investigate the time taken for a model to converge in training accuracy, we can utilise the training QA accuracy over the agent's 200 thousand episodes and compare it to the QAit task's baseline agent's training QA accuracy

Table 2: Agents' results for the QAit task on fixed and random map games. Question Answering (QA) accuracies are shown in black, while the Sufficient Information (SI) scores are blue. Bolded results indicate better testing QA results compared to the other agent.

Map	Model	Location		Existence		Attribute	
		Train	Test	Train	Test	Train	Test
500 Games							
Fixed	GAT-DQN	0.320 (0.320)	0.176 (0.192)	0.706 (0.183)	**0.702** (0.157)	0.674 (0.024)	0.530 (0.019)
	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)
Random	GAT-DQN	0.418 (0.418)	**0.226** (0.230)	0.746 (0.151)	**0.702** (0.129)	0.688 (0.026)	0.476 (0.021)
	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)
Unlimited Games							
Random	GAT-DQN	0.318 (0.318)	**0.212** (0.212)	0.726 (0.152)	**0.708** (0.149)	0.552 (0.026)	**0.512** (0.017)
	DQN	0.316 (0.316)	0.188 (0.188)	0.728 (0.213)	0.668 (0.218)	0.812 (0.055)	0.506 (0.018)

Table 3: Scores of the best performing models on the QAit task for each question type on each map type, across all game settings. Question Answering (QA) accuracies are shown in black, while the Sufficient Information (SI) scores are blue.

Question Type	Agent	Results
Fixed Map		
Location	Rainbow	0.280 (0.280)
Existence	Rainbow	0.692 (0.157)
Attribute	DQN	0.534 (0.014)
Random Map		
Location	Rainbow	0.258 (0.258)
Existence	DDQN	0.694 (0.196)
Attribute	DDQN	0.544 (0.019)

to infer how the GAT aided in the convergence of QA accuracy during training.

6 RESULTS

6.1 Fixed Maps

As can be seen from the results in Table 2, the inclusion of the GAT in the DQN model (GAT-DQN) in 500 fixed map games was not able to aid the DQN agent in achieving better Question Answering (QA) accuracy or Sufficient Information (SI) score for location and attribute type questions. The exception to this is the SI score for attribute types questions which performed higher than the DQN.

GAT-DQN significantly outperformed the DQN in QA accuracy for existence type questions, increasing from 0.674 to 0.702. The GAT-DQN was also able to achieve better results than the best model in the QAit task for this setting. As tabulated in Table 3, the best model for existence type questions in a fixed map game is the Rainbow agent, which achieved a QA and SI score of 0.692 and 0.157, respectively. The GAT-DQN achieved better QA accuracy and matched the SI score.

6.2 Random Map

6.2.1 500 Games. In the 500 random mapped games, we see significant improvements for the location type questions for both QA accuracy and SI score while achieving lower measures in training for both. The GAT-DQN improved the QA accuracy from 0.204 to 0.226 and the SI score from 0.216 to 0.23, respectively. Once again, we can see lower QA accuracies for attribute questions and a higher SI score.

Similarly to fixed maps, we see that the QA accuracy achieved by the GAT-DQN for existence type questions (0.702) was higher than that of the DQN (0.678). Additionally, the GAT-DQN managed to achieve better accuracy than QAit's best model on random maps for existence questions which is a DDQN agent with a QA accuracy of 0.694.

6.2.2 Unlimited Games. The results from the unlimited game experiments (also tabulated in Table 2) indicate an improvement in the GAT-DQN model's ability to answer all three question types compared to the DQN. The increase in training data variability allowed the GAT-DQN to increase the attribute QA accuracy from 0.506 to 0.512, unlike in the 500 game setting. Furthermore, training on unlimited games allowed for better QA accuracy for existence type questions on random maps with an accuracy of 0.708, further improving the best model's accuracy and its SI score from the 500 games model.

7 DISCUSSION

Results from the fixed map experiments show that the training QA accuracy in all three question types was lower in the GAT-DQN than with the DQN, including existence type questions, which performed better on the test set. Being unable to converge to the same training accuracy yet performing better on the test set could suggest the GAT prevented the DQN from overfitting the data or could point to greater generalisability of the model.

Lower training scores for QA and SI can also be seen in location and existence type questions on random maps. Despite these lower training scores, the GAT improved both questions' accuracy, with location only being improved on random maps, implying that

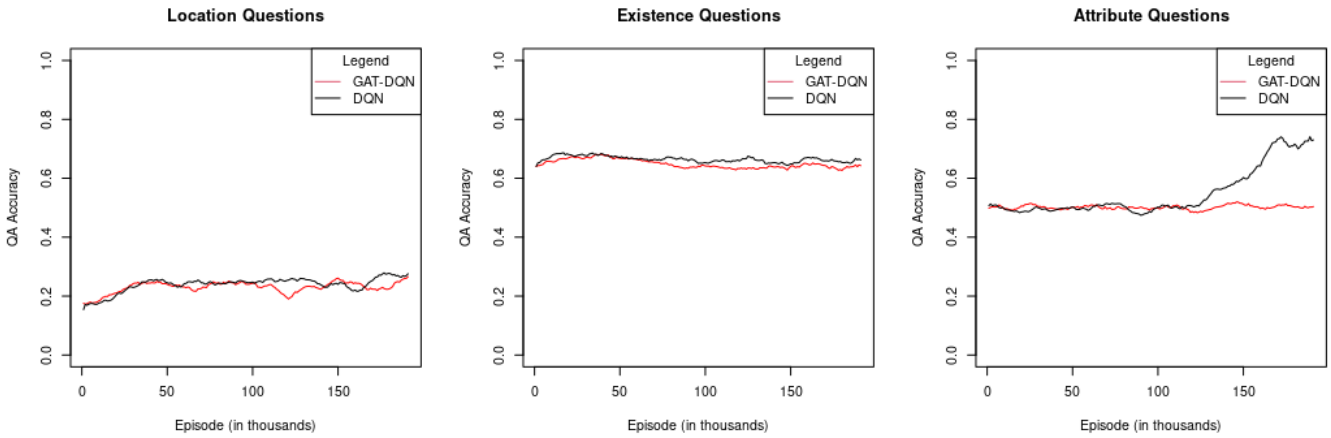

Figure 7: Training QA accuracy using unlimited games on random maps for both the baseline DQN and the GAT-DQN over 200 thousand episodes.

an agent's greater contextual understanding of its surroundings improves generalisability. This postulate is motivated by random map games having randomly sized games. The higher variability within the training and testing sets are thus more challenging to solve and require agents with higher generalising abilities than with fixed mapped games. This relationship can be explored further by looking at the results from the unlimited games, which have maximum variability in training data and negligible possibility of overfitting to the training data. The results suggest that the postulate is correct as unlimited games allowed the GAT to improve the DQN's performance on all three question types, which was not the case on the 500 games setting.

We can also see that although training the GAT-DQN on unlimited games for location type questions improved QA and SI accuracy compared to the regular DQN, it is lower than the GAT-DQN model trained on 500 games. However, we see a more significant increase from the DQN baseline in unlimited games than 500 games, indicating that the higher variability in the training data still resulted in higher utility from the GAT. However, this implies that while the GAT utility increases, it is limited by the agent utilising it as the DQN alone does worse on all three question types when trained on unlimited versus 500 games.

The DQN inherently performs poorly on unlimited games compared to 500 games. However, the GAT bridged the gap in its shortcoming to a marginal degree, as it still could not beat its accuracy from 500 games. This highlights that while evidence suggests that the GAT improves QA accuracy under more generalised environment conditions, the agents' QA accuracy is hindered by its own abilities. The long-term context provided by the GAT improves the QA accuracy but cannot alone solve the task.

7.1 Training Convergence

Figure 7 plots the QA accuracies for the baseline DQN agent and the GAT-DQN for unlimited games on random maps. The complete set of plotted training accuracies for all experiments run can be seen in Figure 8 and Figure 9 in the Appendix. Analysing these curves confirmed the results in Table 2, in that during the training process, the GAT-DQN has slightly lower training QA accuracy in most of the experiments run. In the unlimited game experiments, where GAT-DQN outperformed the DQN in every question type, we can also see slightly lower training QA accuracy throughout the training process. Furthermore, we can see that both the DQN and GAT-DQN have very similar training curves, and there is no evidence to suggest that the GAT aided in the convergence of the model as the training curves start to flatline in the same area for both models across all game settings. The exception is the GAT-DQN on attribute type questions when trained on unlimited games with random map (Figure 7), where the training QA accuracy is significantly lower than its counterpart. However, since it is computationally infeasible to overfit the training data in the unlimited game setting, the GAT could prevent the agent from learning some distribution pattern from the environment sampling that does not yield higher accuracy in unseen environments such as the test set.

8 CONCLUSIONS

This paper demonstrates the use of Graph Attention Networks in aiding an RL agent's ability to answer a question in interactive text-based environments. Results show that while GATs do not aid in the training convergence of the agent, they can increase the question answering accuracy of agents in an IQA setting, provided sufficient training data with high variability.

Better QA results stemming from greater variability in the training data indicate GATs can create more generalisable agents that perform better to unseen environments when answering questions on locality, existence and attributes of objects in text-based environments. However, providing additional context to the agent using a GAT will not solve the task alone and only aid in an agent's performance, suggesting the GAT's performance is limited by the agent utilising it.

9 LIMITATIONS & FUTURE WORK

A limitation the project has is its architecture's inability to process edge labels from the constructed KG. Since a GAT works on a self-attention mechanism over nodes, possible valuable information contained in the edges of the KG (relations between entities) is not utilised, possibly leading to worse performance. An extension to the project could be to implement a particular version of the GAT that includes edge-features [12], which could provide better additional context to an agent.

Due to time and computational constraints, no experimentation was conducted on how the GAT can aid the other RL agents in the QAit baselines, such as the DDQN and the Rainbow agent. These agents are QAit's best agents for particular map and question types, and equipping them with GATs could prove beneficial. Furthermore, running additional experiments on fixed maps with unlimited games could also be beneficial due to the GAT-DQN's observed requirement of higher training data and variability.

Additionally, one could look into using a different approach to KG construction, which could greatly aid in the performance of the model [3]. Instead of using Stanford's OpenIE along with a set of heuristic rules, one could consider a BERT model trained to construct KGs from state descriptions (e.g. Q*BERT [4]). More recently, WorldFormer [3] was developed as another transformer-based KG constructor, which allows QA models to perform significantly better using its constructed KG over Q*BERT's KGs or rule-based approaches.

10 ACKNOWLEDGEMENT

Thank you to my supervisors, Dr Buys and Dr Shock, who were always available when in need of guidance, answering questions, or discussing ideas. Acknowledgement also needs to go to Shane Acton, whose invaluable knowledge on Graph Neural Networks and his availability to answer any questions on Graph Attention Networks was greatly appreciated. My team members, Edan Toledo and Greg Furman, also deserve thanks for their tireless group efforts on the collaborative portion of the project and their overall willingness to discuss ideas for the project at large. Lastly, thank you to the administrators of UCT's High-Performance Cluster for accommodating the rest of my project team and me when requiring greater resources than what was available to us at home.

REFERENCES

[1] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L. Hamilton. 2021. Learning Dynamic Belief Graphs to Generalize on Text-Based Games. arXiv:2002.09127 [cs.CL]

[2] Prithviraj Ammanabrolu and Mark O. Riedl. 2019. Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. arXiv:1812.01628 [cs.CL]

[3] Prithviraj Ammanabrolu and Mark O Riedl. 2021. Learning Knowledge Graph-based World Models of Textual Environments. *arXiv preprint arXiv:2106.09608* (2021).

[4] Prithviraj Ammanabrolu, Ethan Tien, Matthew Hausknecht, and Mark O. Riedl. 2020. How to Avoid Being Eaten by a Grue: Structured Exploration Strategies for Textual Worlds. arXiv:2006.07409 [cs.AI]

[5] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 344–354.

[6] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML]

[8] Dimitri P. Bertsekas. 2005. *Dynamic Programming and Optimal Control* (3rd ed.). Vol. I. Athena Scientific, Belmont, MA, USA.

[9] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 1247–1250.

[10] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. arXiv:1506.02075 [cs.LG]

[11] Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014. Open Question Answering with Weakly Supervised Embedding Models. arXiv:1404.4326 [cs.CL]

[12] Jun Chen and Haopeng Chen. 2021. Edge-Featured Graph Attention Network. arXiv:2101.07671 [cs.LG]

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[14] 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.

[15] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2019. TextWorld: A Learning Environment for Text-based Games. arXiv:1806.11532 [cs.LG]

[16] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851* (2017).

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[18] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. 2016. Towards end-to-end reinforcement learning of dialogue agents for information access. *arXiv preprint arXiv:1609.00777* (2016).

[19] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1608–1618.

[20] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. 2017. A Convolutional Encoder Model for Neural Machine Translation. arXiv:1611.02344 [cs.CL]

[21] 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.

[22] M. Gori, G. Monfardini, and F. Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 2. 729–734 vol. 2. https://doi.org/10.1109/IJCNN.2005.1555942

[23] Jian Guan, Yansen Wang, and Minlie Huang. 2018. Story Ending Generation with Incremental Encoding and Commonsense Knowledge. arXiv:1808.10113 [cs.CL]

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[25] Simon Jegou, Michal Drozdzal, David Vázquez, Adriana Romero, and Y. Bengio. 2017. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. 1175–1183. https://doi.org/10.1109/CVPRW.2017.156

[26] Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education India.

[27] Dan Jurafsky. 2020. *Speech and language processing*. preprint on webpage at https://web.stanford.edu/~jurafsky/slp3/ed3book_dec302020.pdf.

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

[29] Endri Kacupaj, Joan Plepi, Kuldeep Singh, Harsh Thakkar, Jens Lehmann, and Maria Maleshkova. 2021. Conversational Question Answering over Knowledge Graphs with Transformer and Graph Attention Networks. arXiv:2104.01569 [cs.CL]

[30] Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, Washington, USA, 1700–1709. https://www.aclweb.org/anthology/D13-1176

[31] Natalia Konstantinova and Constantin Orasan. 2013. *Interactive Question Answering*. 149 –. https://doi.org/10.4018/978-1-4666-2169-5.ch007

[32] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (March 2019), 452–466. https://doi.org/10.1162/tacl_a_00276

[33] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2017. Gated Graph Sequence Neural Networks. arXiv:1511.05493 [cs.LG]

[34] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 23–33. https://doi.org/10.18653/v1/P17-1003

[35] Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning Dependency-Based Compositional Semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 590–599. https://www.aclweb.org/anthology/P11-1060

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

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

[38] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[39] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[40] Tao Shen, Xiubo Geng, Tao Qin, Daya Guo, Duyu Tang, Nan Duan, Guodong Long, and Daxin Jiang. 2019. Multi-Task Learning for Conversational Question Answering over a Large-Scale Knowledge Base. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 2442–2451. https://doi.org/10.18653/v1/D19-1248

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

[42] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[45] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 9 pages.

[46] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and Tell: A Neural Image Caption Generator. arXiv:1411.4555 [cs.CV]

[47] Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 643–648.

[48] 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).

[49] Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Chris Pal, Yoshua Bengio, and Adam Trischler. 2019. Interactive Language Learning by Question Answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 2796–2813. https://doi.org/10.18653/v1/D19-1280

[50] 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 APPENDIX

A.1 Sufficient Information Score Heuristics for Attribution Questions

Table 4: 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). [49]

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.

A.2 Training Accuracy on Fixed Maps

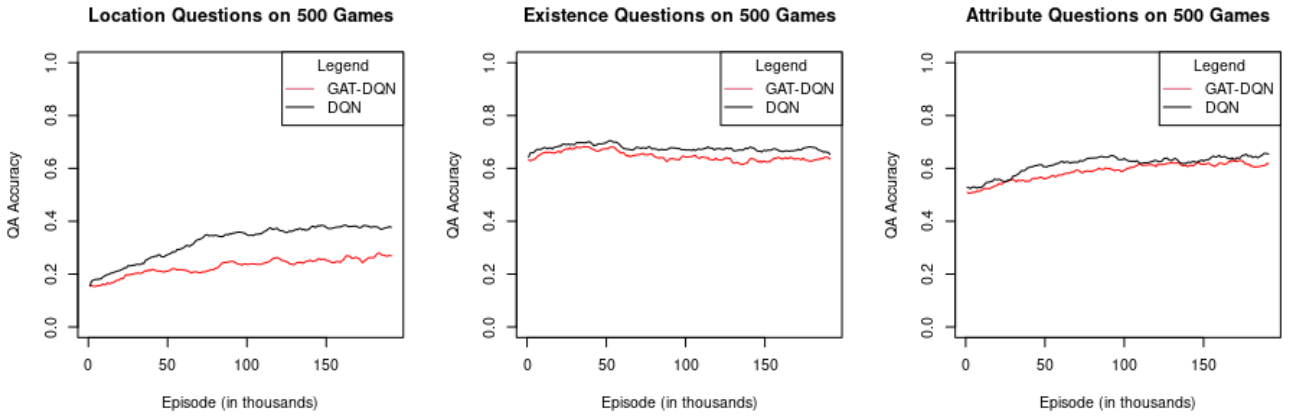

Figure 8: Training QA accuracy on fixed maps for both the baseline DQN and the GAT-DQN over 200 thousand episodes.

A.3 Training Accuracy on Random Maps

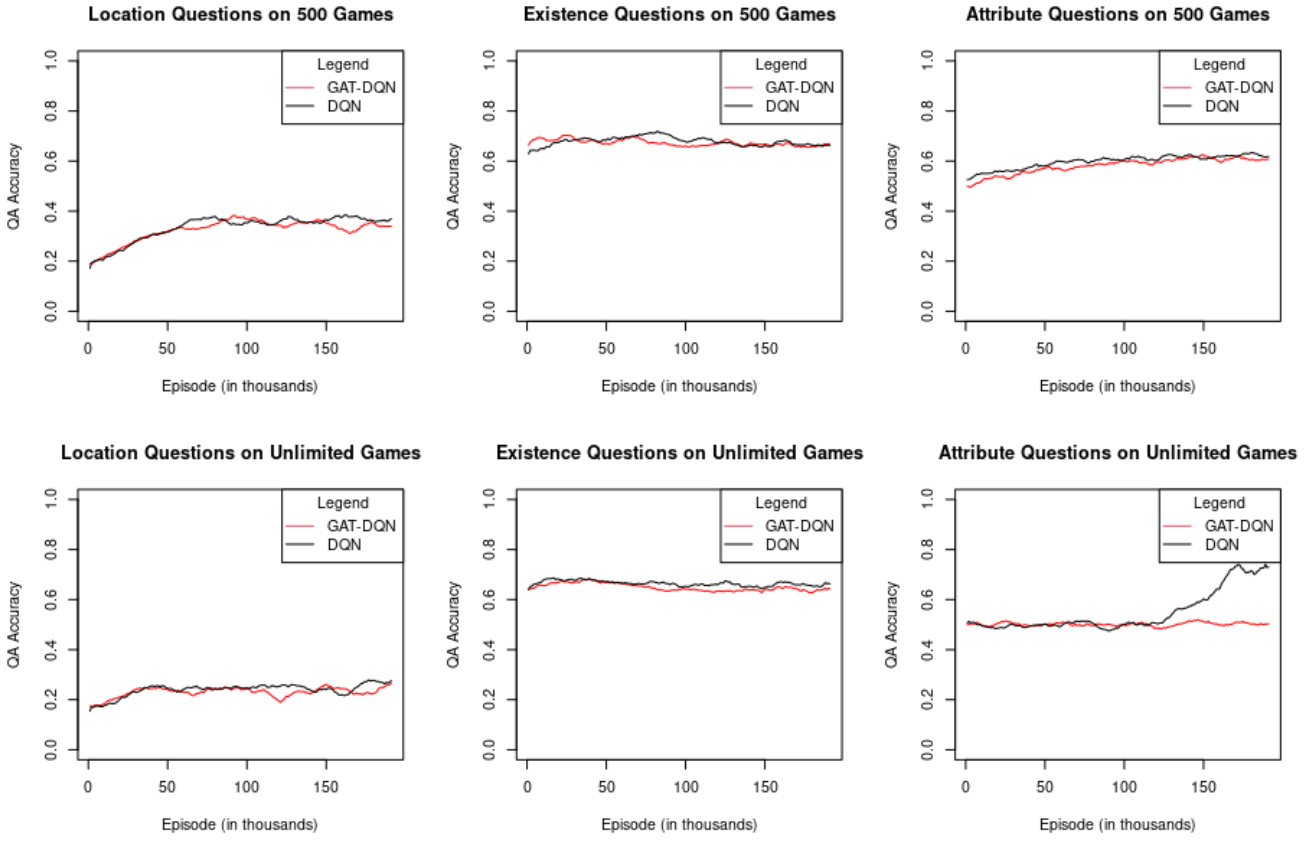

Figure 9: Training QA accuracy on random maps for both the baseline DQN and the GAT-DQN over 200 thousand episodes.